AFRL-RI-RS-TR-2013-057

LOW-COST COGNITIVE ELECTRONICS TECHNOLOGY FOR ENHANCED COMMUNICATIONS AND SITUATIONAL AWARENESS FOR NETWORKS OF SMALL UNMANNED AERIAL VEHICLES (UAV)

VIRGINIA POLYTECHNIC INSTITUTE & STATE UNIVERSITY

MARCH 2013

FINAL TECHNICAL REPORT

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ AIR FORCE MATERIEL COMMAND ■ UNITED STATES AIR FORCE ■ ROME, NY 13441

NOTICE AND SIGNATURE PAGE

REPORT DOCUMENTATION PAGE	*Form Approved* **OMB No. 0704-0188**

1. REPORT DATE *(DD-MM-YYYY)* MARCH 2013	**2. REPORT TYPE** FINAL TECHNICAL REPORT	**3. DATES COVERED** *(From - To)* OCT 2010 – SEP 2012

4. TITLE AND SUBTITLE LOW-COST COGNITIVE ELECTRONICS TECHNOLOGY FOR ENHANCED COMMUNICATIONS AND SITUATIONAL AWARENESS FOR NETWORKS OF SMALL UNMANNED AERIAL VEHICLES (UAV)	**5a. CONTRACT NUMBER** FA8750-11-2-0014
	5b. GRANT NUMBER N/A
	5c. PROGRAM ELEMENT NUMBER 62788F
6. AUTHOR(S) Charles W. Bostian, Alexander R. Young	**5d. PROJECT NUMBER** G2CL
	5e. TASK NUMBER 00
	5f. WORK UNIT NUMBER 01
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Virginia Polytechnic Institute & State University Virginia Tech 1880 Pratt Dr., Ste 2006 Blackburg, VA 24060-6750	**8. PERFORMING ORGANIZATION REPORT NUMBER** N/A
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/RITF 525 Brooks Road Rome NY 13441-4505	**10. SPONSOR/MONITOR'S ACRONYM(S)** N/A
	11. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-RI-RS-TR-2013-057

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited. PA# 88ABW-2013-1071
Date Cleared: 5 MAR 2013

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This document describes the design, construction, and testing of a proof-of-concept hardware and software package that combines cognitive radio and autonomous vehicles in a single system whose behavior captures the essential features of both. The result is a low-cost (less than $200) cognitive radio and cognitive engine package suitable for installation in the small experimental UAVs flown by USAFRL. Experiments with the package controlling a small wheeled vehicle demonstrate its ability to explore and learn a multidimensional environment that combines changing RF, location, and mission data and to optimize its mission performance intelligently.

15. SUBJECT TERMS
cognitive radio, autonomous vehicles, unmanned aerial vehicle, UAV, cognitive engine

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** STEPHEN REICHHART
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	UU	77	**19b. TELEPONE NUMBER (*Include area code*)** N/A

Table of Contents

List of Figures ii

List of Tables iii

1.0 Executive Summary 1

2.0 Introduction 2

2.1 Project Objective 2

2.2 Overview of Cognitive Radio, Autonomous Vehicles, and Combined Cognition 2

2.2.1 Cognitive Radio 2

2.2.2 Autonomous Vehicles 2

2.2.3 Combined Cognition 2

3.0 Methods, Assumptions, Procedures 3

3.1 Operational Assumptions 3

3.2 Use Case 3

3.3 Methods 4

3.3.1 Overview of Experimental Components 4

3.3.2 Experimental Procedure 6

3.3.3 AVEP Details 7

3.3.4 Control Software 23

3.3.5 Node B Details 31

3.3.6 Learning and Decision Making Algorithms 31

4.0 Results and Discussion 45

4.1 Software Simulation 45

4.1.1 Individual Objective Functions 45

4.1.2 Multi-domain Decision Making in a Static Environment 46

4.1.3 Multi-domain Decision Making in a Simple Dynamic Environment 51

4.1.4 Multi-domain Decision Making in a Highly Dynamic Environment 53

4.2 Live Tests 56

4.2.1 Live Multi-domain Decision Making in a Static Environment 57

4.2.2 Live Multi-domain Decision Making in a Simple Dynamic Environment 60

4.2.3 Live Multi-domain Decision Making in a Highly Dynamic Environment 62

5.0 Conclusions 65

6.0 Recommendations 65

7.0 References 66

8.0 List of Acronyms 69

List of Figures

Figure 1. Autonomous vehicle experiment platform. 4
Figure 2. Photograph of AVEP test bed. 5
Figure 3. The central aspect of test bed graph is three paths (edges) that diverge from the decision point at Node 1 and converge again at Node 2. 5
Figure 4. Autonomous vehicle experiment platform. (front view) 8
Figure 5. BeagleBoard-xM single board computer. 9
Figure 6. Hope RF RFM22B FSK RFIC. 9
Figure 7. CW power delivered to a 50 ohm load as a function of carrier frequency for a stock 433 MHz RFM22B and for the same module with the output networks removed. 10
Figure 8. RFM22B reference implementation [20]. 11
Figure 9. RFM22B transmit filter network. 11
Figure 10. RFM22B transmit filter network with bypass. The dotted line show the implemented short circuit used to bypass the filter network. 11
Figure 11. RSSI values as a function of CW RF power at the antenna terminal. 12
Figure 12. Receiver's passband characteristics. 13
Figure 13. Sample of memory registers set and read on the RFM22B [1]. 13
Figure 14. BeagleBoard-xM trainer board. 15
Figure 15. SKIRL radio package, showing BeagleBoard-xM, trainer board, and RFM22B. 15
Figure 16. SKIRL schematic showing individual components along with their functions. 16
Figure 17. NXT Brick and chassis. 16
Figure 18. Conceptual AVEP system architecture showing components and process flow. 18
Figure 19. High-layer software organization of AVEP system. 19
Figure 20. AVEP radio software stack. 20
Figure 21. Radio software stack compared to SKIRL hardware components. 21
Figure 22. AVEP controller finite state machine. 23
Figure 23. RF subsystem packet structure. 26
Figure 24. Organization of flags field in RF subsystem packet. This figure shows the flags available to a Node A radio. 27
Figure 25. The test bed graph contains three separate paths between Node 1 and Node 2. 29
Figure 26. A more complicated test bed layout with additional nodes and edges, and their attendant paths. 30
Figure 27. Organization of flags field in RF subsystem packet. This figure shows the flags available to a Node B radio for communication with a Node A radio. 31
Figure 28. The ODA loop applied to CR and AV scenarios. 32
Figure 29. The cognition cycle used by the AVEP. 33
Figure 30. Graphical representation of Z objective function as a function of targets and anti-targets. 36
Figure 31. Experimentally recorded values of time (to travel a fixed-length path) and requisite AVEP rotor power values, with 3rd order polynomial fit. 37
Figure 32. Graphical representation of T objective function as a function of path distance and rotor power. 38
Figure 33. BER waterfall curve for noncoherent FSK. 40
Figure 34. E_b/N_0 values as a function of SNR and R_s. 40
Figure 35. Graphical representation of B objective function as a function of SNR and R_s. 41
Figure 36. Graphical representation of G objective function as a function of time and bit rate. 42
Figure 37. AVEP in operation during a live test experiment. 57

List of Tables

Table 1. Path length (in meters) of each path in the test bed. 5
Table 2. Rotor power and time measurements for AVEP, used to determine AVEP velocity. 37
Table 3. Environmental parameters (meters) used in evaluating UMDDM in a static environment. 47
Table 4. Results of UMDDM decision making simulation in a static environment over 20 iterations. 48
Table 5. Solutions generated by decision maker on iterations 4, 5, and 10, and the scores associated with each solution. 48
Table 6. Parameters associated with solutions shown in Table 6. 48
Table 7. Results of UMDDM decision making simulation in a static environment with N = −82 dBm. 50
Table 8. Solutions generated by decision maker for simulated environment where N = −82 dBm. 50
Table 9. Parameters associated with solutions shown in Table 8 51
Table 10. Environmental parameters (meters) used in evaluating UMDDM in a simple dynamic environment. 51
Table 11. Results of UMDDM decision making simulation in a simple dynamic environment 52
Table 12. Solutions generated by decision maker in a simple dynamic environment, and the scores associated with each solution. 52
Table 13. Parameters associated with solutions shown in Table 12. 53
Table 14. Environmental parameters (meters) used in evaluating UMDDM in a highly dynamic environment. 54
Table 15. Results of UMDDM decision making simulation in a highly dynamic environment over 10 iterations. 55
Table 16. Solutions generated by decision maker in a highly dynamic environment, and the scores associated with each solution. 56
Table 17. Parameters associated with solutions shown in Table 16. 56
Table 18. Environmental parameters (meters) used in evaluating live UMDDM in a static environment. 57
Table 19. Results of live UMDDM in a static environment over 10 iterations. 58
Table 20. Solutions generated by decision maker during live tests in a static environment. 58
Table 21. Parameters associated with solutions shown in Table 20 59
Table 22. Results of live UMDDM in a static environment over 10 iterations using higher noise floor. 59
Table 23. Solutions generated by decision maker during live tests in a static environment with higher noise floor. 60
Table 24. Parameters associated with solutions shown in Table 23. 60
Table 25. Environmental parameters (meters) used in evaluating UMDDM in a simple dynamic environment during a live experiment. 60
Table 26. Results of live UMDDM in a simple dynamic environment over 10 iterations in a live experiment. 61
Table 27. Solutions generated by decision maker during live tests in a simple dynamic environment. 61
Table 28. Parameters associated with solutions shown in Table 27. 62
Table 29. Environmental parameters (meters) used in evaluating UMDDM in a highly dynamic environment during a live experiment. 63
Table 30. Results of live UMDDM in a simple dynamic environment over 10 iterations in a live experiment. 64
Table 31. Solutions generated by decision maker in a highly dynamic environment during a live experiment. 64
Table 32. Parameters associated with solutions shown in Table 31. 65

1.0 Executive Summary

This document describes an implementation of the Virginia Tech (VT) cognitive engine on a BeagleBoard-xM single board computer based on the Texas Instruments (TI) OMAP 3530 processor and its successful use to provide simultaneous intelligent control of a frequency-agile and mode-agile radio and an autonomous vehicle. This provides a proof-of-concept prototype of a cognitive system that is aware of its environment, its user's needs, and the rules governing its operation. This prototype should be able to take intelligent action based on this awareness to optimize its performance across both the mobility and radio domains while learning from experience and responding intelligently to ongoing environmental and mission changes. It combines the key features of cognitive radios and autonomous vehicles into a single package whose behavior integrates the essential aspects of both.

The use case for this work is a scenario where a small unpiloted aerial vehicle (UAV) is traversing a nominally cyclic or repeating flight path (an "orbit") seeking to observe targets and where possible avoid hostile fire. As the UAV traverses the path, it experiences varying radio frequency (RF) effects, including multipath propagation and terrain shadowing. The goal is to provide the capability for the UAV to learn the flight path with respect both to motion and RF characteristics and modify radio parameters and flight characteristics proactively to optimize performance. Using sensor fusion techniques to develop situational awareness, the UAV should be able to adapt its motion or communication based on knowledge of (but not limited to) physical location, antenna orientation, radio performance, and channel conditions. Using sensor information from RF and MOT (MOT short for physical MOTion) domains, the UAV uses the mission objectives and its knowledge of the world, to decide on a course of action. The UAV develops and executes a multi-domain action; action that crosses domains, such as changing the RF power and increasing its speed.

We present in detail the design of a low-cost (less than $200) package called SKIRL based on the BeagleBoard-XM computer and the Hope RF RFM22B RF integrated circuit that is suitable for installation in the small experimental UAVs flown by the Air Force Research Laboratory (AFRL). In the work documented here, SKIRL is integrated with a set of target, navigational, and environmental sensors mounted on a LEGO wheeled vehicle that executes a hypothetical two-dimensional mission based on the UAV use case while avoiding the costs and potential security problems associated with a flight test. Experiments with the system demonstrate its ability to explore and learn a multidimensional environment that combines changing RF, location, and mission data and to optimize its mission performance intelligently. So far as we are aware, this is the first successful demonstration of its kind.

Beginning with a review of the literature of cognitive radio and autonomous vehicles, we discuss the rationale for combining the two technologies and move through the practical steps of designing, building, and testing a prototype. We show how the architecture of a typical cognitive radio (consisting of a cognitive engine and a programmable RF unit) can be expanded to include the sensors and actuators associated with an autonomous vehicles and provide the software and hardware details necessary for implementation. This includes possibly the first development of a low-cost cognitive radio platform based on a low-cost RF integrated circuit instead of a software defined radio. We explore the issues associated with testing and evaluating a cognitive device and develop an appropriate test procedure for the prototype considered here. The test results clearly demonstrate that the vehicle is capable of exploring and learning a complex environment and meeting the intended objectives. We recommend testing in an airborne platform as a next step.

2.0 Introduction

2.1 Project Objective

The primary objectives of this project are to implement the VT cognitive engine CSERE (Cognitive System Enabling Radio Evolution) on a Beagleboard or BeagleBoard-xM single board computer based on the Texas Instruments (TI) OMAP 3530 processor. (See http://beagleboard.org/.) This implementation coupled with a frequency- and mode-agile radio will provide a prototype flight-capable integrated cognitive engine and cognitive radio suitable for testing on an autonomous vehicle.

2.2 Overview of Cognitive Radio, Autonomous Vehicles, and Combined Cognition

2.2.1 Cognitive Radio

A cognitive radio (CR) is a transceiver that is aware of its environment, its own capabilities, the rules within which it can operate, and its operator's needs and privileges. It is capable of changing its operating modes in ways that achieve the user's objectives and maximize performance while staying within the rules. A radio that does these things is said to be adaptive. If it is capable of learning in the process of changing its operating modes and of developing configurations that its designer never anticipated, it is said to be cognitive. The semantic difference between adaptive and cognitive is subject to argument and immaterial to this discussion. Conceptually a cognitive radio consists of a frequency and mode-agile radio (usually a software defined radio, SDR) platform controlled by a cognitive engine (CE), an artificial intelligence software package that serves as the system's brain, "turning the knobs" and "reading the meters" of the radio platform. It follows the OODA loop well known to the USAF. For a detailed overview, see [1].

2.2.2 Autonomous Vehicles

The field of autonomous vehicles can be traced back to the seminal work "Cybernetics " by Norbert Weiner and, by association, the Macy Conferences [2]. An autonomous vehicle is a vehicle capable of independent and autonomous motion that can react in an intelligent manner to changes in its environment. Conner provides a thorough philosophical discussion of autonomous vehicles as well as a good overview of the history of robotics and autonomous vehicles through 2000 [3].

2.2.3 Combined Cognition

Combined cognition is the term we use to cover areas where autonomous vehicle aspects and cognitive radio functionality are combined. The concept of combining CR and AV technology seemed to originate with Hauris [66], who simply recast the work of Rieser et al. [4], [5] in an AV scenario. And while Hauris specifically addresses optimization of RF parameters for a "geographically varying wireless network" in the introduction, no effort is made to consider geographic information (i.e. position) in the optimization process.

Angermann, Frassl and Lichtenstern take a step closer to CR/AV convergence with a communication relay chain based on quadrocopter micro air vehicles (MAVs) [6]. A central node calculates position for each MAV node based on simple geometry, dividing the end-to-end link into equally spaced relays based on the number of nodes present. The central node pushes the position information out to each node, who attempt to perform station keeping without any onboard positioning information. The non-distributed nature of the decision making limits the potential of the system, as does the lack of onboard positioning on each node.

Communication-aware motion (CAM) is where we start to see CR and AV systems come together. CAM refers to mobile nodes that are capable of taking communication parameters into account when planning and executing motion. Mostofi notes that “a mobile network that is deployed in an outdoor environment can experience uncertainty in communication, navigation and sensing.” In this situation, “optimum motion-planning decisions considering only sensing and navigation may not be the best for communication, resulting in communication and sensing tradeoffs,” [7]. While Mostofi presents a method of using noise variance, signal-to-noise ratio (SNR), and statistical channel models to plan and adapt motion, Mostofi limits his work to one perspective. His observation quoted above opens the door to much more: the possible benefits of adapting RF operation as well.

The other perspective, the opposite side of CAM, is position/motion-aware communication (P/MAC). And there is very little research in this area. Amanna et al. briefly allude to P/MAC but limit possibilities to matters of policy [8], that which is required by regulations. Di Felice et al. also seem to flirt with the idea of P/MAC, in the form of geolocation and national spectrum databases that vehicles can access in order to more easily perform vehicle-based DSA [9]. But again, Di Felice et al. seem to miss the potential that exists; the concept of extending location based decision-making beyond DSA completely lost.

3.0 Methods, Assumptions, Procedures

3.1 Operational Assumptions

Our work focuses on small UAV platforms flown by AFRL. While the subjects of cognitive radio and cooperative communications both have a rather vast and mathematically complicated literature frequently based on idealized assumptions, we have concentrated on what we think can be achieved in practice using near-term available hardware and software that is compatible with the small UAV platforms of interest.

The USAF is understandably disinclined to provide detailed specifications about its UAV programs. Lacking formal specifications from AFRL, we developed a reasonable use case to guide the development of this project. Additionally, we created a mission objective function that we feel reflects reasonable and possible mission considerations for an AFRL UAV. Finally we developed an indoor test bed that maps to recent tests conducted by AFRL.

In this project we have treated the mobile radio node as the master node or node A in a master/slave radio setup. A base station is the node B or slave in this master/slave radio setup. While this differs from cell terminology in which the base station is the node A and the mobile unit is the node B, this makes sense as in our research, the mobile node is the controller, directing the base station actions.

While the mobile radio platform is capable of frequency agility across a wide frequency band, in our research the radio was constrained to a single transmitting frequency, allowing us to observe how other parameters changed in response to variability in the environment.

3.2 Use Case

The research deals with a scenario where a UAV is flying a nominally cyclic or repeating flight path. As the UAV traverses the path, it experiences varying RF effects, including multipath propagation and terrain shadowing. The goal is to provide the capability for the UAV to learn the flight path with respect to motion and RF characteristics and modify radio parameters and/or motion behavior proactively to mitigate deleterious effects. Using sensor fusion techniques to develop situational awareness, the UAV should be able to adapt its motion or communication based on knowledge of (but not limited to) physical location, antenna orientation, radio performance, and channel conditions. Using sensor information from RF and MOT (MOT for physical motion) domains, the UAV uses the mission objectives and its knowledge of the world, to decide on a course of action. The UAV develops and executes a multi-domain action; action that crosses domains, such as changing the RF power and increasing its speed.

3.3 Methods

3.3.1 Overview of Experimental Components

3.3.1.1 Autonomous Vehicle Experiment Platform (AVEP)

The research in this dissertation includes software algorithms as well as a physical component; a robotic/RF hybrid system that implements the software algorithms in a laboratory environment. The AVEP, shown in Figure 1 is the hardware and software platform that we developed to deploy the unified multi-domain decision making (UMDDM) algorithms. The AVEP consists of two distinct physical subsystems, the computational platform and the chassis. The chassis provides the mobility capability, while the computational platform runs all system and control software, as well as the algorithms developed for this research.

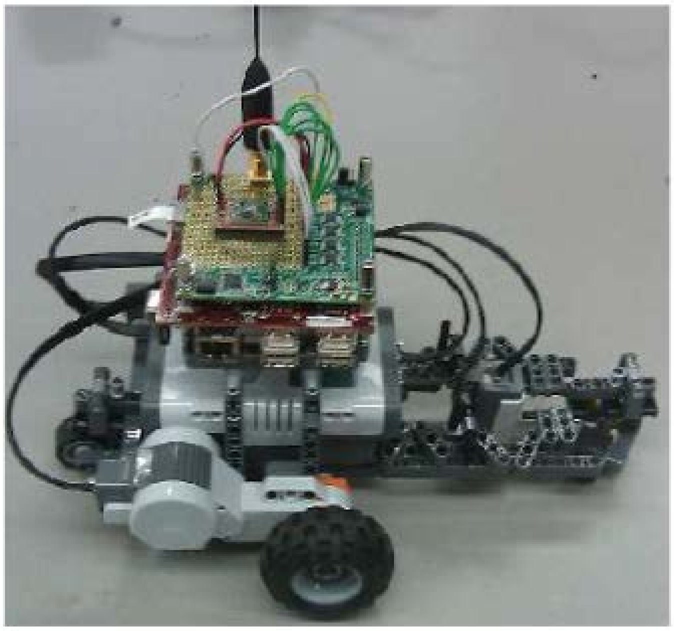


Figure 1. Autonomous vehicle experiment platform.

3.3.1.2 Node B Radio (NBR)

While the AVEP consists of an RF component and a MOT component, the NBR consists simply of an RF component, as a standalone radio. However, it is the radio system with which the AVEP communicates when the AVEP is employing its RF capabilities.

3.3.1.3AVEP Test Bed

The AVEP test bed is used as the testing ground for the physical AVEP implementation. The test bed represents a single section of the UAV flight path; one critical point where the AVEP must make a choice. A picture of the test bed is shown in Figure 2. The central aspect of the test bed (Figure 3) is three paths that diverge from the decision point at Node 1 and converge again at Node 2. Both Node 1 and Node 2 have barcodes marking their position, and a barcode reader is used by the AVEP to determine position, as discussed in Sections 4.4.3.1 and 5.2.2.3.

The test bed has been designed so that each path has a different length, enough of a length difference to ensure that travel along each path by the AVEP is a different experience. Table 1 lists the path length (in meters) of each path in the test bed.

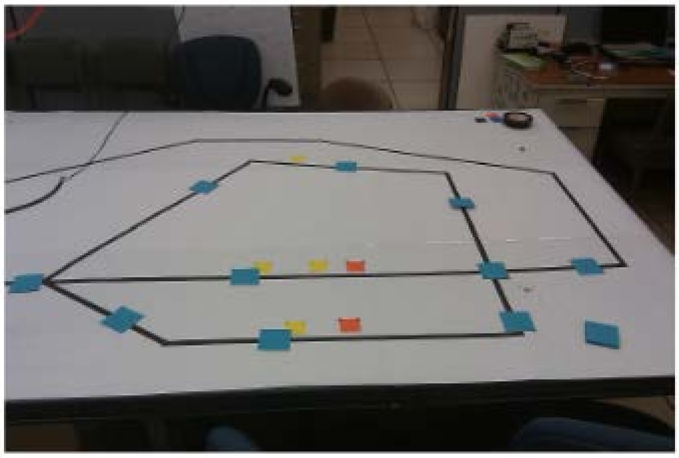


Figure 2. Photograph of AVEP test bed.

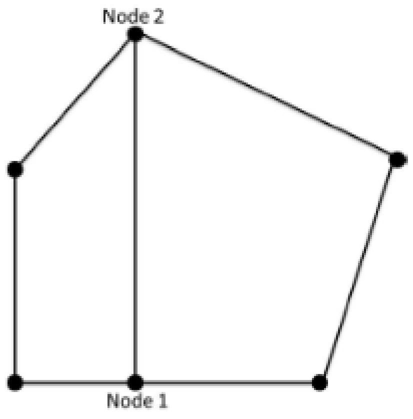


Figure 3. The central aspect of test bed graph is three paths (edges) that diverge from the decision point at Node 1 and converge again at Node 2.

Table 1. Path length (in meters) of each path in the test bed.

Path	Length (m)
1	1.575
2	1.219
3	2.223

3.3.2 Experimental Procedure

The experimental procedure has been developed to specifically target RF and MOT parameters. The experimental procedure consists of two major components: software simulation, and live testing. The core of the system is the decision making component. The decision maker uses data from the system's sensor to generate solutions that the system uses to implement action. The simulation is a software analysis of the system's core decision making component. For the live system testing, we built a robotic component, called the AVEP. The decision making code that is analyzed in the simulation is implemented as part of the AVEP, which is then deployed on the test bed described above in Section 6.3.1.3.

3.3.2.1 Evaluation of Test Results

Evaluation of the test results is the method by which researchers validate their research. In research that deals with intelligence and decision making, this can be very tricky. According to Bartholomew, "Human intelligence is one of the most important yet controversial topics in the whole field of the human sciences. It is not even agreed whether it can be measured or, if it can, whether it should be measured," [10]. And yet, still engineers and scientists take great efforts to validate research. In order to validate our research, we can look to other research in similar areas for evaluation methods.

3.3.2.1.1 CR Performance

Traditionally the performance of a CR has been measured using various figures of merit including throughput/goodput, network stability, and spectrum utilization [5], [11], [12]. These metrics are very appealing due to their familiarity to RF engineers, and their representation of real RF channel and network characteristics. However, many CR researchers are interested not only in RF metrics, but in the performance of the cognition process itself, the ability to determine whether a given solution applied during the cognition process is in fact suitable to the present scenario. Zhao et al. provide an overview of performance metrics, "from node-level to network-level and application level." While they also used a CR testbed for evaluating the process of selecting, incorporating, and evaluating performance metrics and utility functions, they limited themselves to traditional RF metrics: QoS, spectral efficiency, power efficiency, and time of solution adaptation [13]. Dietrich, Wolfe, and Vanhoy have proposed a method of evaluating CR performance using psychometric approaches, i.e., psychological measurement models that are used in testing of humans, [14]. Kaminski has proposed the use of measure of effectiveness (MoEs) two evaluate CR performance using a two stage algorithm based on neural networks [15].

3.3.2.1.2 AV Performance

AV performance is primarily evaluated an AV's ability to perform its intended task. Techy et al. note "UAVs hold the desired altitude with high precision and converge to a phase synchronized state" in [16]. The rules for the DUC state that vehicles must demonstrate the ability to "Complete a mission defined by an ordered series of checkpoints in a complex route network," while observing standard traffic etiquette and regulations [17]. The rules for the 2012 RoboCup state simply "The team scoring the greater number of goals during a match is the winner." [18]. DARPA has recently announced a new robotics challenge seeking to foster innovative research in the area of robotics for disaster response. The broad agency announcement (BAA) indicates that the scoring criteria and competition rules "have not yet been defined," but that candidate scoring criteria includes "successful event completion, completion time, data rate, and energy consumption" [19].

3.3.2.1.3 AVEP Performance

Clearly, evaluating the performance of the AVEP is not straight forward. CR researchers have proposed methods of evaluating the cognitive aspects of CRs, but there are no standards, nor are there even fully implemented prototypes. Methods for evaluating performance of AVs focus primarily on whether the AV in question can successfully complete its assigned task. In light of this, we have followed the example set by DARPA in the DARPA Robotics Challenge (DRC). We will evaluate the AVEP on a variety of factors, including mission success and completion time. Additionally, we will consider two common RF metrics: bit error rate (BER) and packet delivery.

3.3.2.2 Experimental Data

Data is at the heart of the decision making process. Sensor data provides the input to the decision maker. The decision maker uses several objective functions in its decision making process. The individual objective functions represent both the RF and MOT domains, and are:

- Z, Target/Anti-target Value,
- T, Time,
- B, Bit Error Rate, and
- G, Packet Delivery.

In this system, we wish to minimize the bit error rate and the time of travel along a path, and maximize the packet delivery and target/anti-target value. We use a nondominated sort to determine the final solution [Z, T, B, G] that the AVEP will implement. The decision making process, and the objective functions and their supporting algorithms are discussed in depth in Section 6.3.6.

3.3.3 AVEP Details

3.3.3.1 Hardware Platform

The AVEP, shown in Figure 4 is the hardware and software platform that we implemented to deploy and test the decision making and learning algorithms developed in this research. It is a robotic line follower with integrated RF communication system. The AVEP is composed of two distinct physical subsystems, the SKIRL computational platform, and the NXT brick and chassis. SKIRL is the major computational component of the system, while the NXT brick and chassis enable AVEP MOT capability and select environmental sensing capabilities. The AVEP components are discussed in greater detail below.

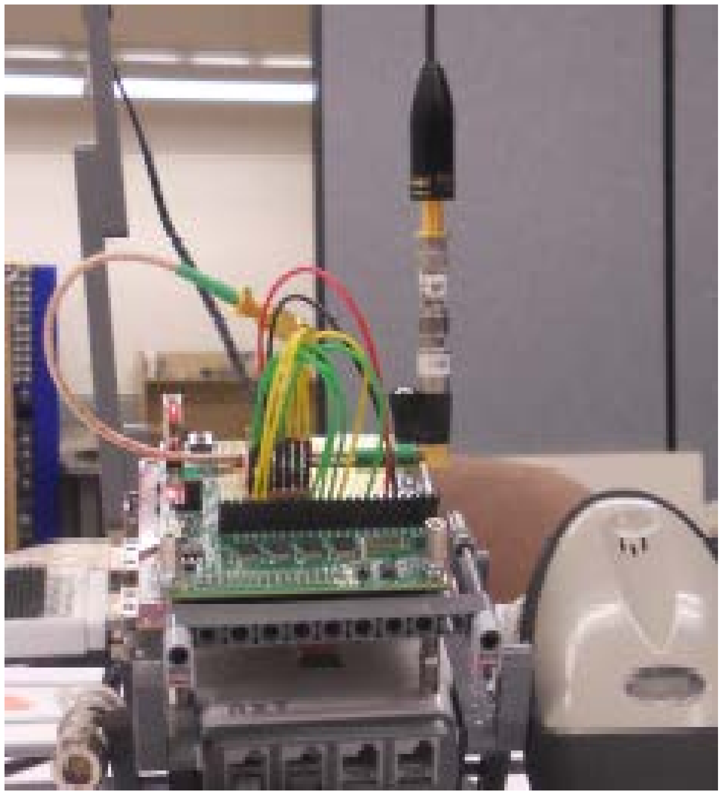


Figure 4. Autonomous vehicle experiment platform. (front view)

3.3.3.1.1 Physical Components: SKIRL Radio Platform

As noted above, SKIRL is the major computational component of the system, and is responsible for running all the control software, as well as radio operation. SKIRL comprises three components discussed below: BeagleBoard-xM single board computer, Hope RF RFM22B RFIC, and trainer board.

3.3.3.1.1.1 BeagleBoard-xM

The BeagleBoard-xM shown in Figure 5 is a low-power, low-cost, single-board computer, using Texas Instruments components. The BeagleBoard-xM features a TI DM3730 with 1 GHz ARM Cortex A8, onboard Ethernet and four port USB hub, with a small footprint (8.26 cm x 8.26 cm). Designed and built to target mobile and video applications, the BeagleBoard-xM supports the addition of expansion boards that can provide additional functionality. The BeagleBoard-xM fully supports Linux and the attendant free and open source tool chains and development environments such as gcc, Python, Emacs, Subversion, etc. The BeagleBoard-xM also has many I/O ports available for interfacing with other components, using serial peripheral interface (SPI) or inter-integrated circuit (I2C), in addition to standard general purpose I/O (GPIO).

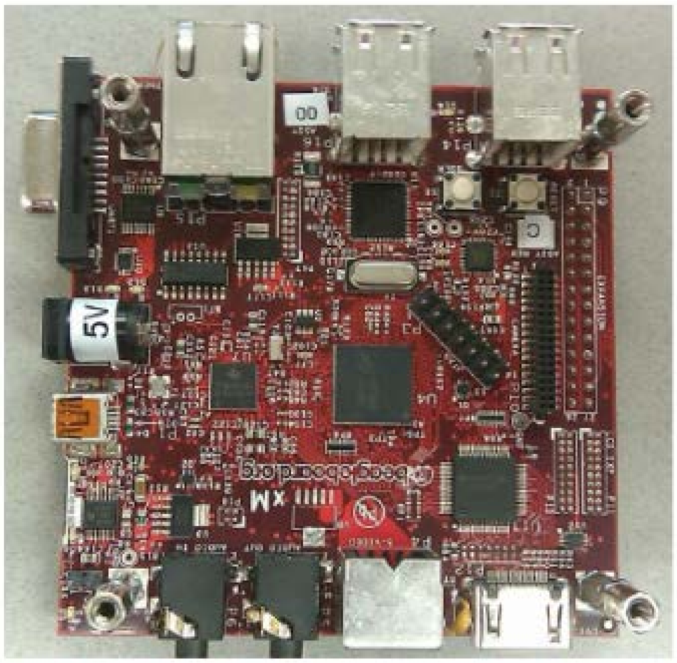


Figure 5. BeagleBoard-xM single board computer.

3.3.3.1.1.2 Hope RF RFM22B

The Hope RF RFM22B [20] shown in Figure 6 is a low cost highly configurable transceiver capable of transmitting and receiving FSK, Gaussian frequency shift keying (GFSK), and on-off keying (OOK) waveforms between 240 MHz and 930 MHz with a maximum transmitter output power of +13 dBm. Advertised receiver sensitivity ranges from -121 dBm to -101dBm, depending on data rate and modulation used. The module is marketed as a fully contained radio solution, providing all the necessary mixing, filtering, tuning, and A/D components to transmit and receive packetized bit data.

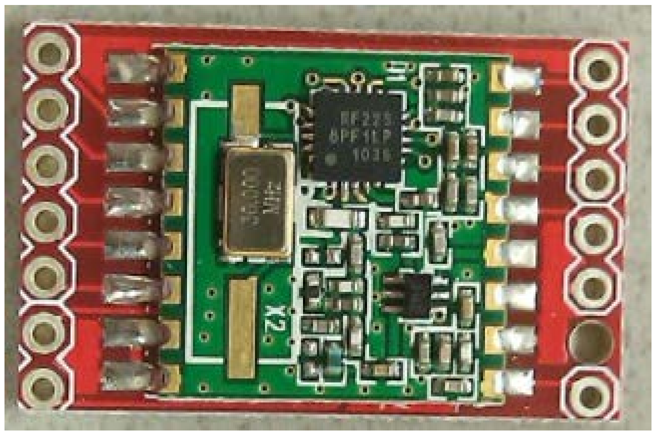


Figure 6. Hope RF RFM22B FSK RFIC.

While the data sheet might be read to imply continuous 240 MHz – 930 MHz, the RFM22B is sold in three versions designed for operation in the 433, 868, and 915 MHz bands. These differ in their output impedance matching and filter networks. For experimental transceivers operating in closed RF environments, removing the output networks allows satisfactory operation over a continuous frequency range 250 to 900 MHz. See Figure 7 for an illustration of the transmitter frequency coverage with and without modification.

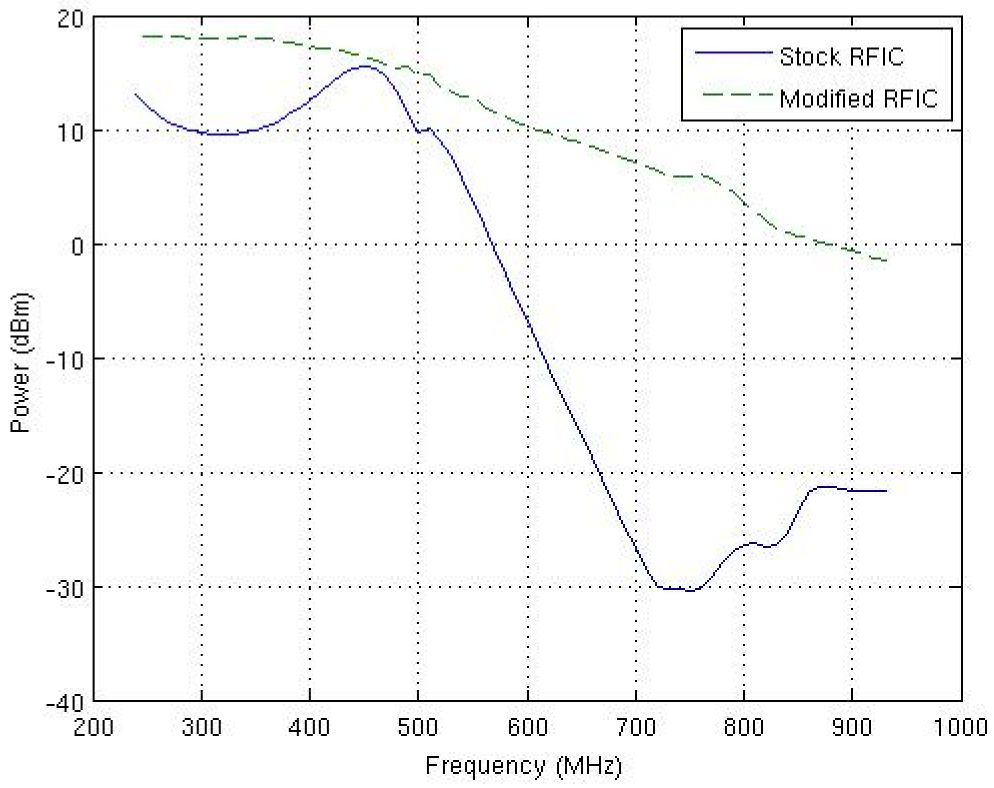


Figure 7. CW power delivered to a 50 ohm load as a function of carrier frequency for a stock 433 MHz RFM22B and for the same module with the output networks removed.

The RFIC contains a filter network between the microcontroller unit (MCU) and the transmit/receive (T/R) switch. Figure 8 presents the reference design of the RFM22B as presented in the data sheet [20]. However, our experience indicates that the filter implementation on-chip actually looks like the network shown in Figure 9, and the network can be bypassed (Figure 10) to increase the transmit power to the level shown in Figure 7.

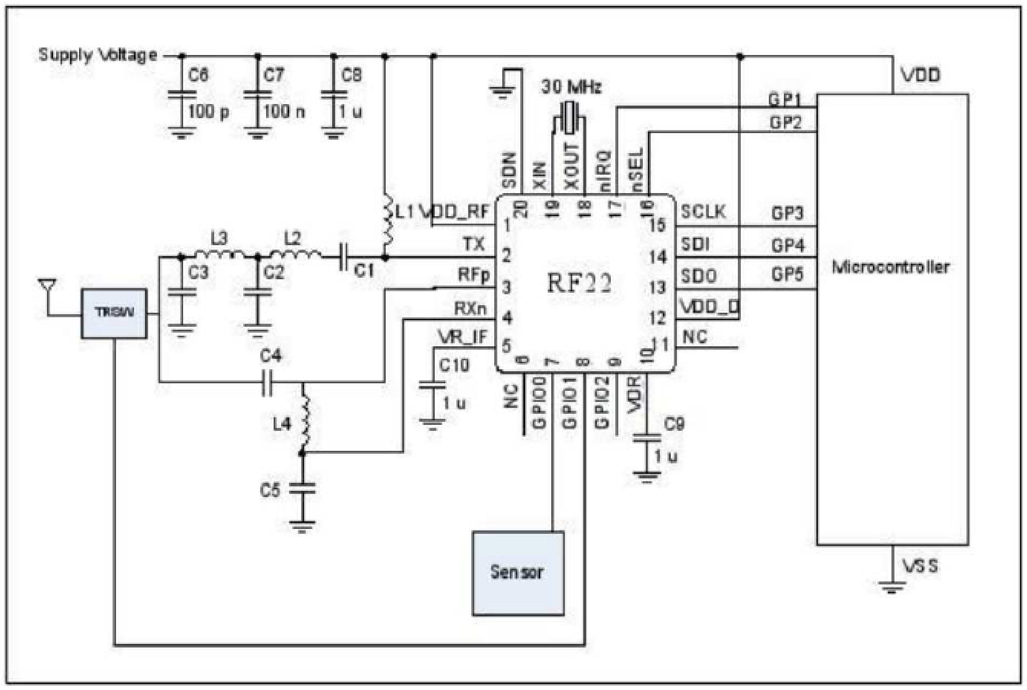


Figure 8. RFM22B reference implementation [20].

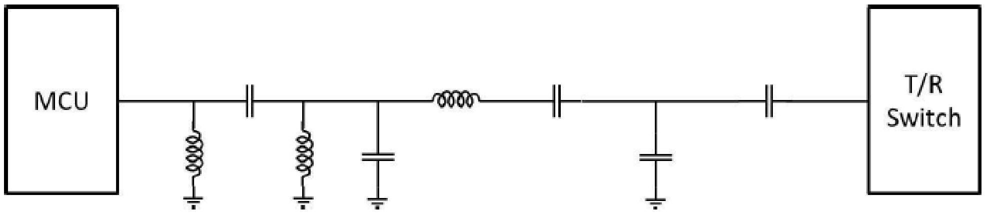


Figure 9. RFM22B transmit filter network.

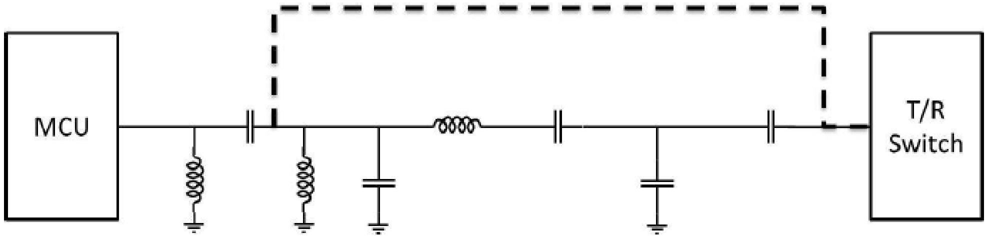


Figure 10. RFM22B transmit filter network with bypass. The dotted line show the implemented short circuit used to bypass the filter network.

The RFM22Bs receiving characteristics of interest are shown in Figure 11 and Figure 12. The received signal strength indicator(RSSI) —which is used to sense whether a channel is occupied or clear—has a useful range extending from about -60 dBm to -10 dBm and the effective RF bandwidth for measuring RF energy in a channel is about 100 kHz.

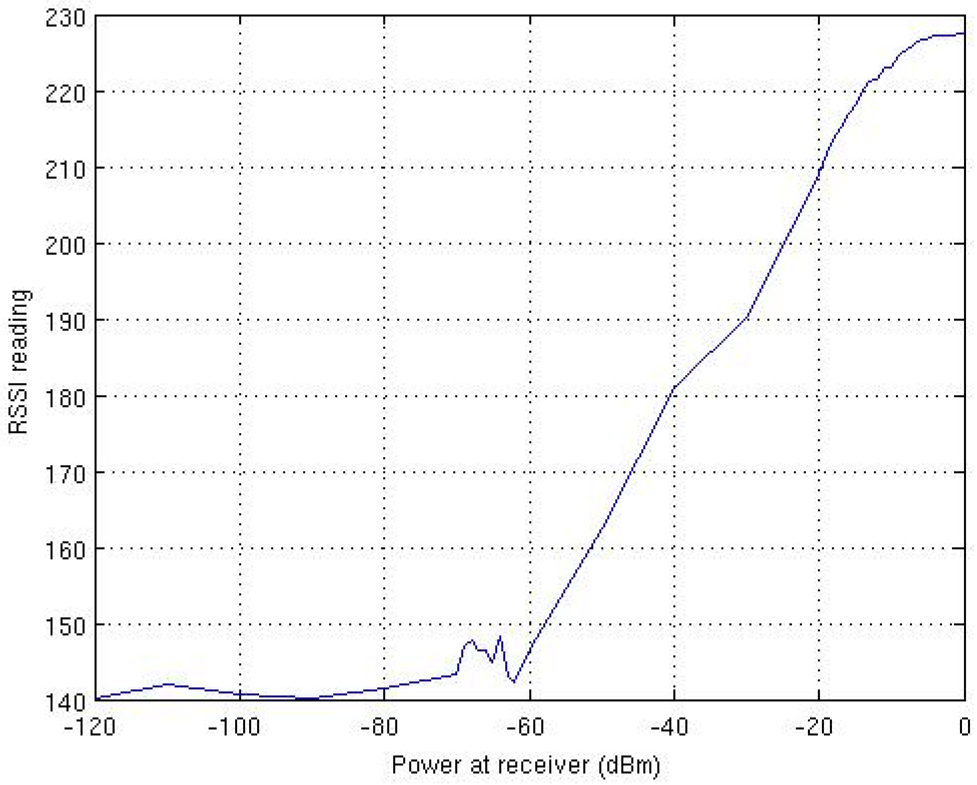


Figure 11. RSSI values as a function of CW RF power at the antenna terminal.

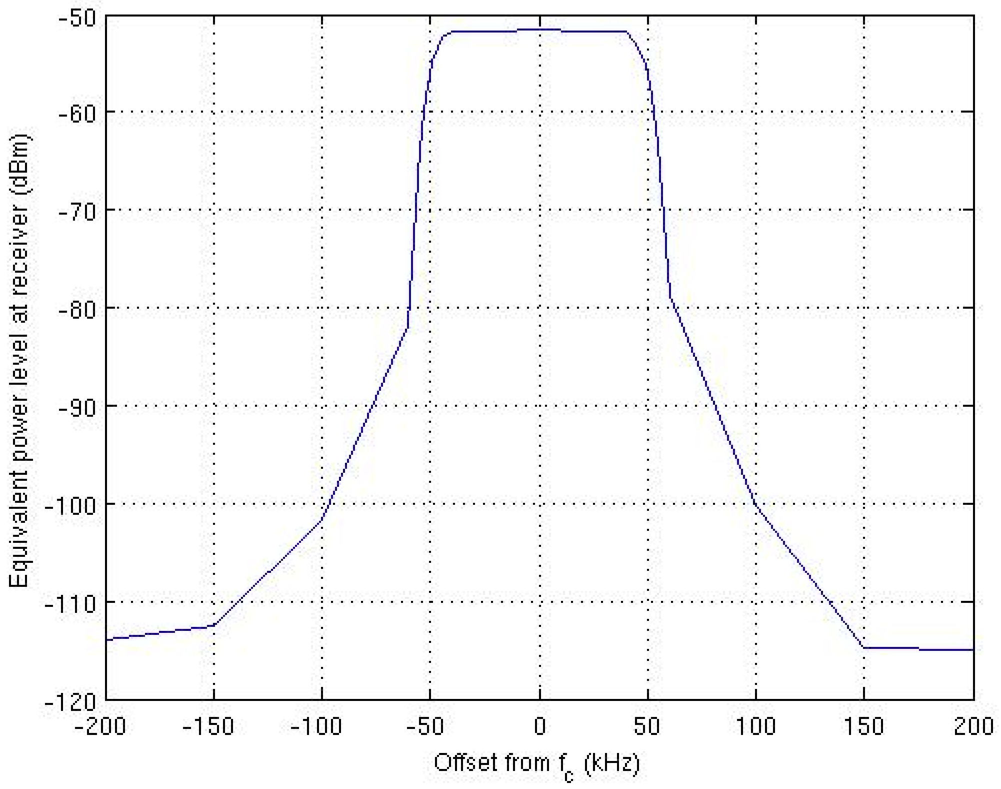
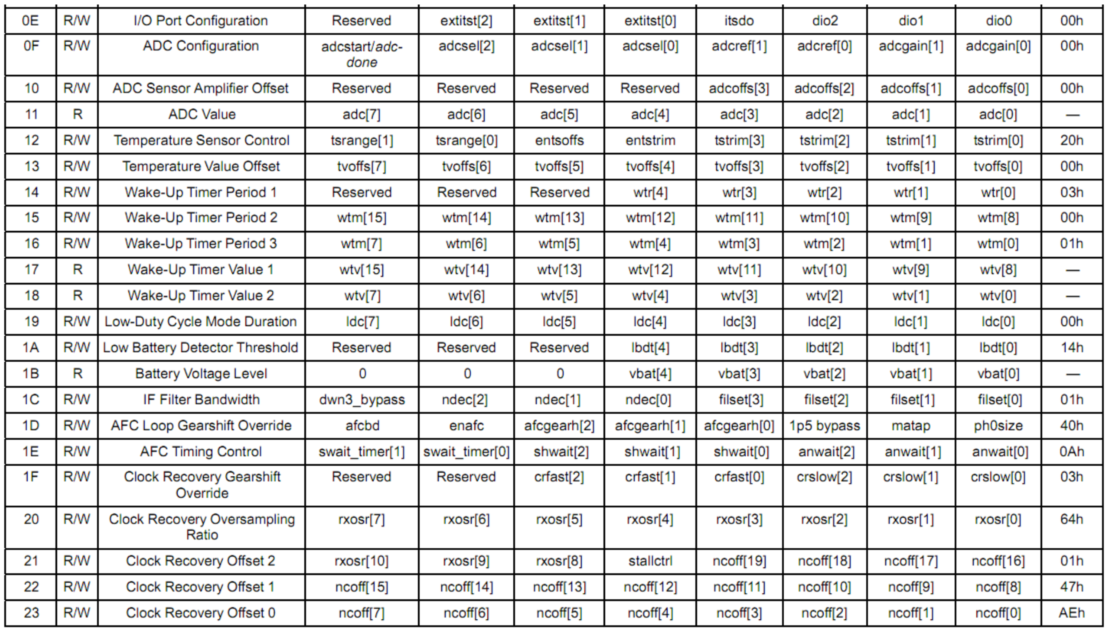


Figure 12. Receiver's passband characteristics.



0E	R/W	I/O Port Configuration	Reserved	extitst[2]	extitst[1]	extitst[0]	itsdo	dio2	dio1	dio0	00h
0F	R/W	ADC Configuration	adcstart/*adc-done*	adcsel[2]	adcsel[1]	adcsel[0]	adcref[1]	adcref[0]	adcgain[1]	adcgain[0]	00h
10	R/W	ADC Sensor Amplifier Offset	Reserved	Reserved	Reserved	Reserved	adcoffs[3]	adcoffs[2]	adcoffs[1]	adcoffs[0]	00h
11	R	ADC Value	adc[7]	adc[6]	adc[5]	adc[4]	adc[3]	adc[2]	adc[1]	adc[0]	—
12	R/W	Temperature Sensor Control	tsrange[1]	tsrange[0]	entsoffs	entstrim	tstrim[3]	tstrim[2]	tstrim[1]	tstrim[0]	20h
13	R/W	Temperature Value Offset	tvoffs[7]	tvoffs[6]	tvoffs[5]	tvoffs[4]	tvoffs[3]	tvoffs[2]	tvoffs[1]	tvoffs[0]	00h
14	R/W	Wake-Up Timer Period 1	Reserved	Reserved	Reserved	wtr[4]	wtr[3]	wtr[2]	wtr[1]	wtr[0]	03h
15	R/W	Wake-Up Timer Period 2	wtm[15]	wtm[14]	wtm[13]	wtm[12]	wtm[11]	wtm[10]	wtm[9]	wtm[8]	00h
16	R/W	Wake-Up Timer Period 3	wtm[7]	wtm[6]	wtm[5]	wtm[4]	wtm[3]	wtm[2]	wtm[1]	wtm[0]	01h
17	R	Wake-Up Timer Value 1	wtv[15]	wtv[14]	wtv[13]	wtv[12]	wtv[11]	wtv[10]	wtv[9]	wtv[8]	—
18	R	Wake-Up Timer Value 2	wtv[7]	wtv[6]	wtv[5]	wtv[4]	wtv[3]	wtv[2]	wtv[1]	wtv[0]	—
19	R/W	Low-Duty Cycle Mode Duration	ldc[7]	ldc[6]	ldc[5]	ldc[4]	ldc[3]	ldc[2]	ldc[1]	ldc[0]	00h
1A	R/W	Low Battery Detector Threshold	Reserved	Reserved	Reserved	lbdt[4]	lbdt[3]	lbdt[2]	lbdt[1]	lbdt[0]	14h
1B	R	Battery Voltage Level	0	0	0	vbat[4]	vbat[3]	vbat[2]	vbat[1]	vbat[0]	—
1C	R/W	IF Filter Bandwidth	dwn3_bypass	ndec[2]	ndec[1]	ndec[0]	filset[3]	filset[2]	filset[1]	filset[0]	01h
1D	R/W	AFC Loop Gearshift Override	afcbd	enafc	afcgearh[2]	afcgearh[1]	afcgearh[0]	1p5 bypass	matap	ph0size	40h
1E	R/W	AFC Timing Control	swait_timer[1]	swait_timer[0]	shwait[2]	shwait[1]	shwait[0]	anwait[2]	anwait[1]	anwait[0]	0Ah
1F	R/W	Clock Recovery Gearshift Override	Reserved	Reserved	crfast[2]	crfast[1]	crfast[0]	crslow[2]	crslow[1]	crslow[0]	03h
20	R/W	Clock Recovery Oversampling Ratio	rxosr[7]	rxosr[6]	rxosr[5]	rxosr[4]	rxosr[3]	rxosr[2]	rxosr[1]	rxosr[0]	64h
21	R/W	Clock Recovery Offset 2	rxosr[10]	rxosr[9]	rxosr[8]	stallctrl	ncoff[19]	ncoff[18]	ncoff[17]	ncoff[16]	01h
22	R/W	Clock Recovery Offset 1	ncoff[15]	ncoff[14]	ncoff[13]	ncoff[12]	ncoff[11]	ncoff[10]	ncoff[9]	ncoff[8]	47h
23	R/W	Clock Recovery Offset 0	ncoff[7]	ncoff[6]	ncoff[5]	ncoff[4]	ncoff[3]	ncoff[2]	ncoff[1]	ncoff[0]	AEh

Figure 13. Sample of memory registers set and read on the RFM22B [1].

The module’s operation is configured (and reconfigured) by values stored in its internal registers. Figure 13 shows some of the memory registers on the RFM22B that can be set to configure the radio operation (and subsequently read to determine what the current configuration is).

As an example of how the radio is controlled, the two lines below set the data rate by writing values (txdr1 and txdr0) to the upper and lower transmit data rate registers TX Data Rate 1 and TX Data Rate 0. The data rate value is a 16 bit value, and TX Data Rate 1 holds the upper 8 bits of the value, while TX Data Rate 0 holds the lower 8 bits.

```
self._set_reg_tx_rate_1(txdr1)

self._set_reg_tx_rate_0(txdr0)
```

The primary I/O mechanism for the RF module is a first in first out (FIFO) buffer. Serial data bytes are written to the transmit (TX) FIFO buffer in succession for transmission.

When the buffer is full, the accumulated bytes are transmitted. Received data is likewise stored serially in the receive (RX) FIFO buffer. Continued reads will transfer all the data out of the buffer to the user. From the user perspective, it appears that the TX and RX FIFO buffers are one and the same, as they are both accessed through writing to and reading from the same register address, however there are in fact two FIFO buffers, one for TX and one for RX and internal RFIC controls ensure proper access to the appropriate FIFO.

An SPI bus is the primary method of interaction with the RFM22B, and four SPI lines are used to send and receive data to and from the module. GPIO is used for secondary signaling; controlling a T/R switch and providing a path for reading hardware interrupts.

3.3.3.1.1.3 Trainer Board

There are minor issues to be resolved in interfacing the radio platform and the BeagleBoard-xM. The trainer board shown in Figure 14 and available from Tin Can Tools [21] solves these problems while providing access to SPI, I2C, and GPIO interfaces, a circuit prototyping area, and an onboard ATMEL ATmega328 processor. It provides level translators that convert the 1.8 V signals from the BeagleBoard-xM to 3.3 V for serial communication with the radio module, and it converts the radio module signals from 3.3 V to 1.8 V for serial communication in the opposite direction.

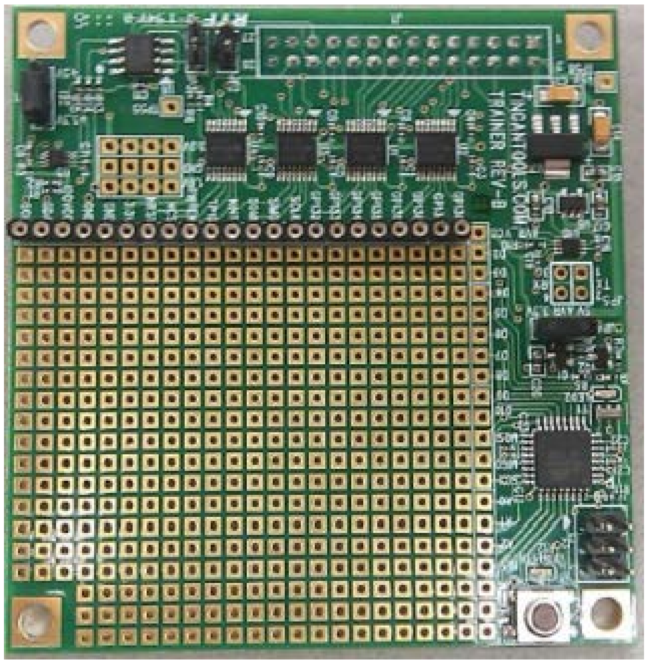


Figure 14. BeagleBoard-xM trainer board.

3.3.3.1.2 *SKIRL Integration*

Fully integrated, the SKIRL radio platform components stack to make a single package, as shown in Figure 15. The system schematic in Figure 16 shows the components and their functions. The radio module provides FSK-based radio communications and the trainer board provides logic level translation for serial communications. The BeagleBoard-xM contains a Linux kernel and Ubuntu operating system. The BeagleBoard-xM also contains the software that operates the radio. While all the radio operations actually take place on the radio module itself, the software on the BeagleBoard-xM initializes the radio module and controls its operation by reading values from and writing values to the radio module's registers. Communication between the radio module and the BeagleBoard-xM is achieved using four SPI communication lines, and three GPIO lines. In addition to the signaling lines, the BeagleBoard-xM also provides power and ground to the trainer board and—indirectly via the trainer board—to the radio module.

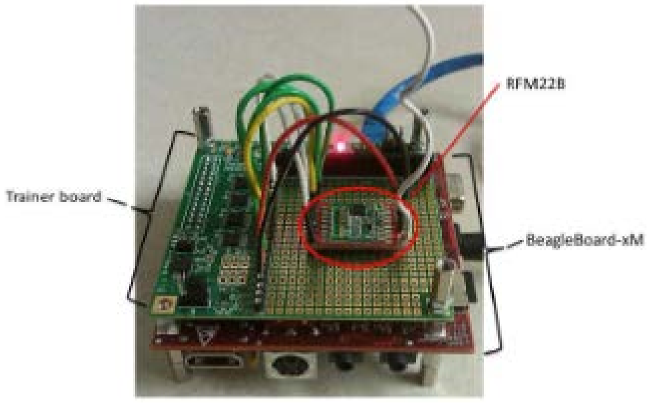


Figure 15. SKIRL radio package, showing BeagleBoard-xM, trainer board, and RFM22B.

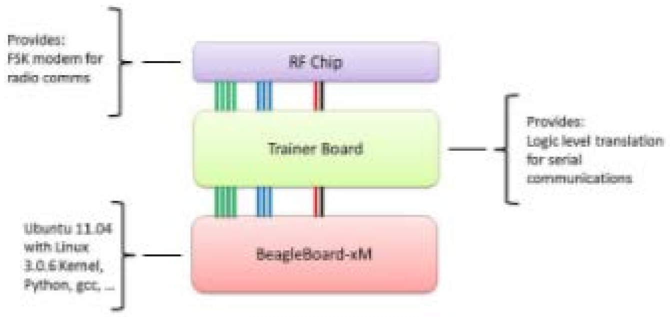

Provides: FSK modem for radio comms
RF Chip
Trainer Board
Provides: Logic level translation for serial communications
Ubuntu 11.04 with Linux 3.0.6 Kernel, Python, gcc, ...
BeagleBoard-xM

Figure 16. SKIRL schematic showing individual components along with their functions.

3.3.3.1.3 *Physical Components: NXT Brick and Chassis*

The NXT brick and chassis form (Figure 17) the motion platform of the AVEP. Both the brick and the chassis come from the Lego Mindstorms NXT robotics toolkit [22]. The chassis features wheels and rotors for locomotion, as well as various sensors used by the system. The NXT brick forms the structural core of the chassis; the rotors, sensors, and Lego interconnect pieces all connect to the NXT brick, and the SKIRL platform rests physically on the brick.

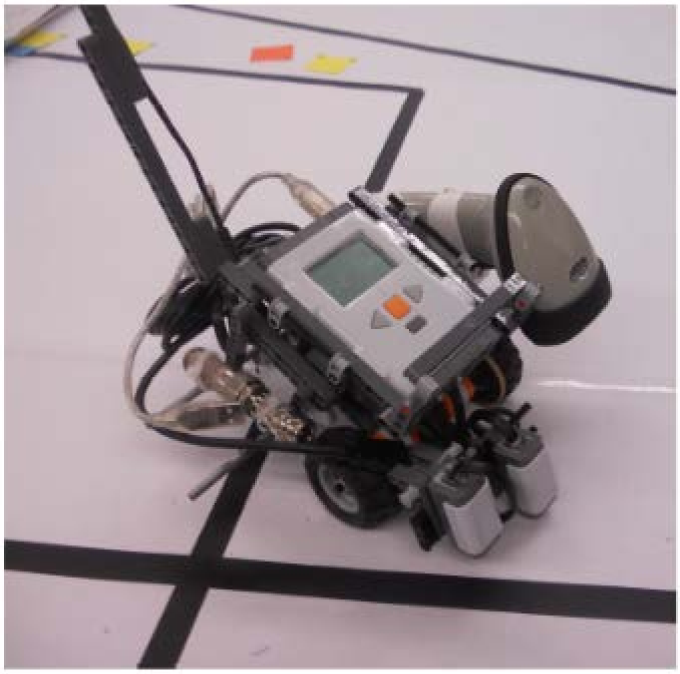


Figure 17. NXT Brick and chassis.

The NXT brick contains an ARM processor and integrated LCD for display. The brick contains three rotor connector ports and four sensor connector ports. The ARM processor on the brick natively supports two languages (the graphical NXT-G language and NI's LabVIEW for Lego Mindstorms), and with minimal effort is capable of running programs written in many more languages.

3.3.3.1.3.1 Sensors

The chassis currently carries three sensors: a light sensor, color sensor, and barcode reader. The light and color sensors are NXT native sensors, and connect to the brick. The light sensor responds to reflected light and measures the intensity of the reflected light. This functionality is used in the AVEP line following algorithm. The color sensor also responds to reflected light, but in contrast to the light sensor, the color sensor analyzes the reflected light to determine the light color. The color sensor is used in the AVEP target/anti-target detection algorithm. Unlike the light and color sensors, the barcode reader is not an NXT native sensor, nor is it connected to the NXT brick. While the barcode sensor is mounted on the chassis, it connects directly to the SKIRL radio platform. The NXT sensors were chosen due to their capability to integrate with the NXT brick, and for their relatively low cost.

The barcode reader is used in the position/location algorithm. The barcode was chosen as the method of position/location determination due to its simplicity and low cost. The testbed is set up indoors, and GPS works very poorly (if at all) indoors. As well, standard GPS does not have sufficient resolution to be useful at the scale used by the testbed. Other indoor positioning systems that use infrared [23] are prohibitively expensive for this research. The barcode is a hand held laser scanner similar to those used in retail point-of-sale systems. The barcode scanner has a trigger button to activate the laser, but for this research, we configured the laser to remain on continuously, allowing the scanner to automatically read any barcode that the laser passes over. Modern smart phones such as the iPhone or Android base phones can use their integrated cameras to take a picture of a barcode and use image recognition software to read the value embedded in the barcode image. However, we found the time to take a picture and analyze the barcode took too long to be practical; while the hand held scanner can read and decode a barcode in > 1 second, the smart phone method often took up to 5 seconds to return a result.

3.3.3.2 System Software

3.3.3.2.1 Software Options and Choices

The recommended operating system for the BeagleBoard-xM is Angstrom [24], and it is used in the board validation process [25]. Angstrom is configured using the Open Embedded (OE) embedded development framework [26]. OE allows users to deploy a full operating environment for the BeagleBoard-xM, including operating system (OS) and tool chains. OE builds the operating environment on a separate system, cross compiled for the BeagleBoard-xM platform. For a more detailed discussion of development for the BeagleBoard, see [27]. OE allows completely custom distribution builds for the BeagleBoard, at the expense of having to build every aspect of that custom distribution. Alternatives to Angstrom and OE include OS and tools from the Linaro project [28] and the Ubuntu ARM project [29]. Ubuntu has a history of stable operation and a large selection of available packages. As our laboratory uses Ubuntu on all research and development workstations, we decided to leverage the wealth of institutional knowledge with regards to system installation and support and chose the Ubuntu ARM as the OS for the SKIRL computational platform.

For cognitive engine code and control software for SKIRL, Python [30] was the obvious choice for us. Python is easy to read and code, and Python applications are readily ported to multiple platforms. As with other choices, our institutional knowledge of Python helped make the choice; the CSERE cognitive engine [31] is implemented entirely in Python, and GNU Radio [32] makes heavy use of Python as well.

3.3.3.2.2 System Architecture

The concept for the AVEP system architecture is presented in Figure 18. The system architecture is inspired by the system architectures developed and implemented by DGC and DUC competitors.

Figure 19 shows the software organization of the AVEP in actual implementation. The controller is computational and functional core of the system, and is responsible for starting the radio and motion software, as well as integrating the cognition, route planning, positioning, and other sensor information.

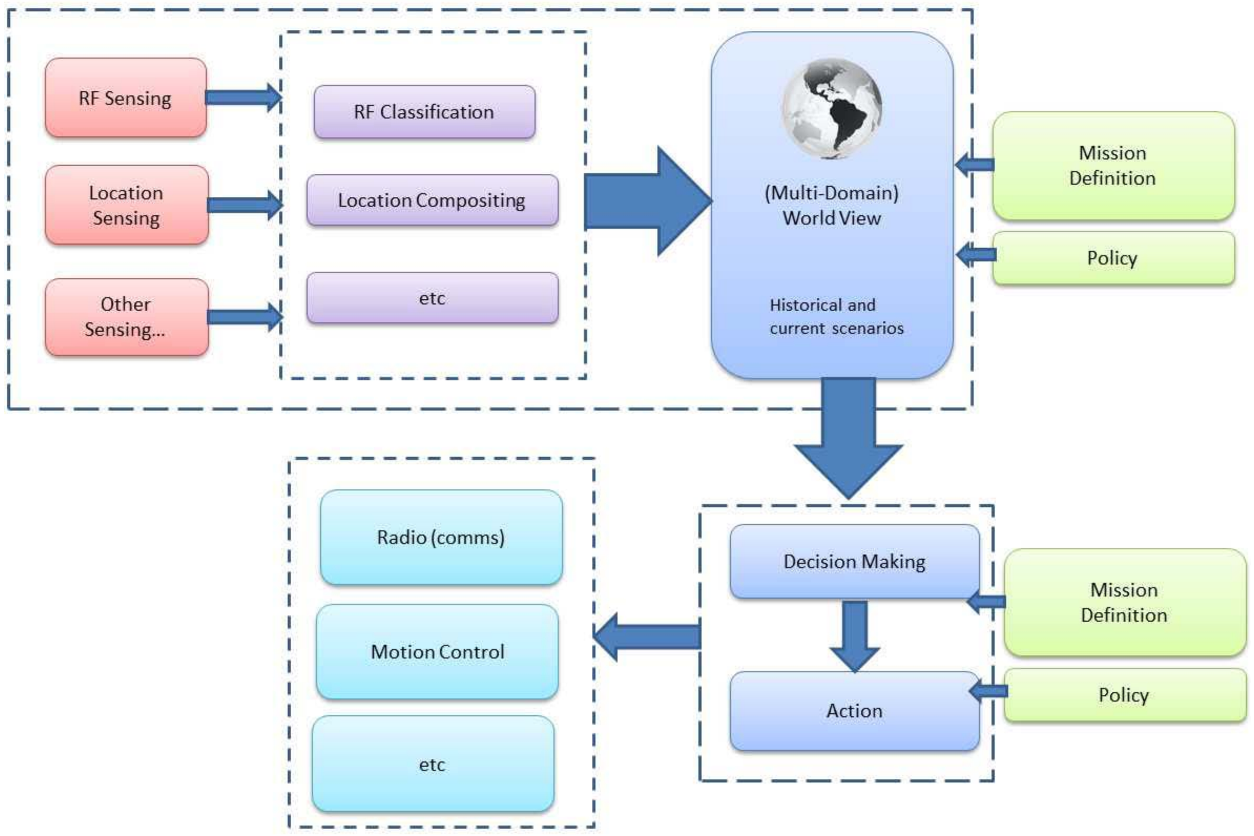


Figure 18. Conceptual AVEP system architecture showing components and process flow.

All system software for SKIRL runs on the BeagleBoard-xM. While the microcontroller on the trainer board is capable of hosting and running programs written in C or assembly, the microcontroller is currently unused. The radio module itself is effectively a closed system; while we can interact with it at some level using its readable and writable registers; its internal operation is hidden. The AVEP system is written entirely in Python. Python provides cross-platform functionality and is easy to read and code, with its similarity to pseudo-code.

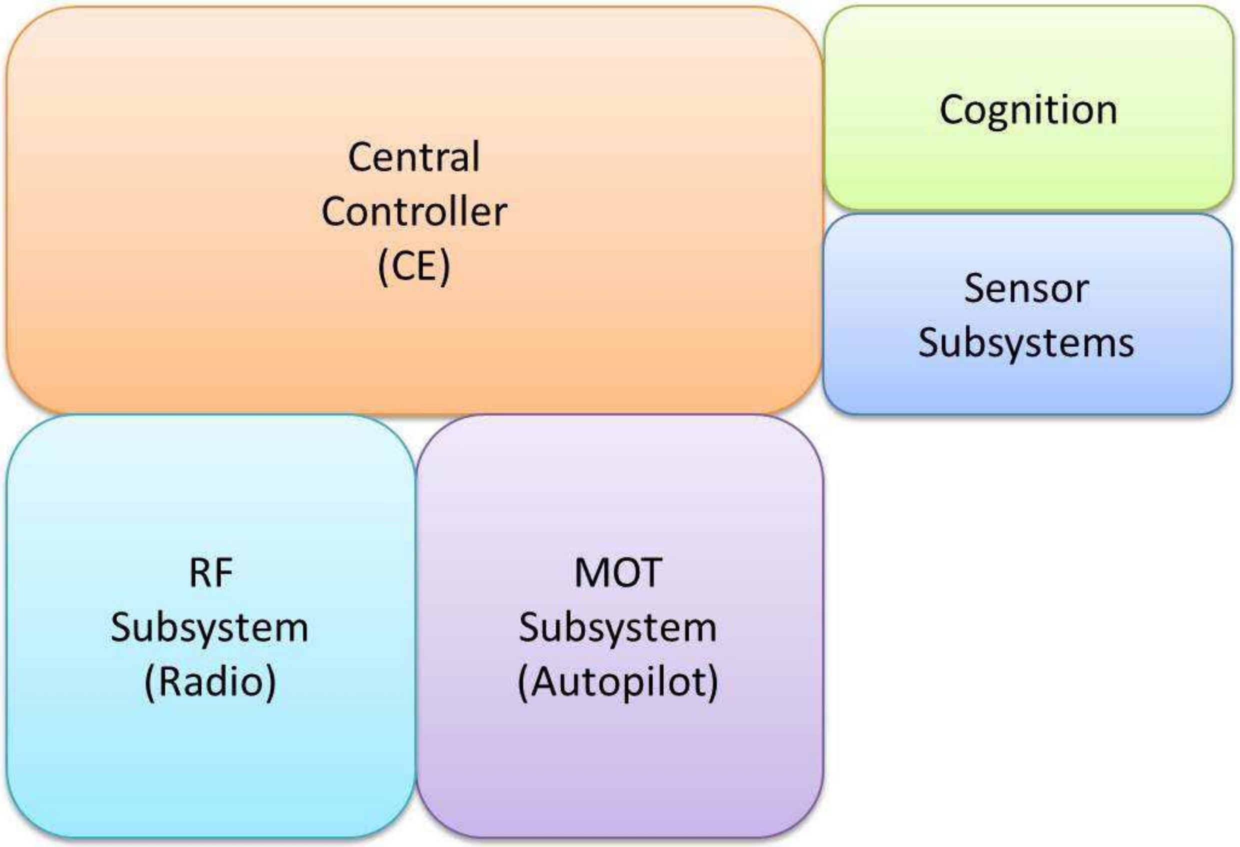


Figure 19. High-layer software organization of AVEP system.

3.3.3.2.3 Radio Software

The radio software is written and organized as a stack (Figure 20), with each layer providing functionality to the layer above and built on the layer below.

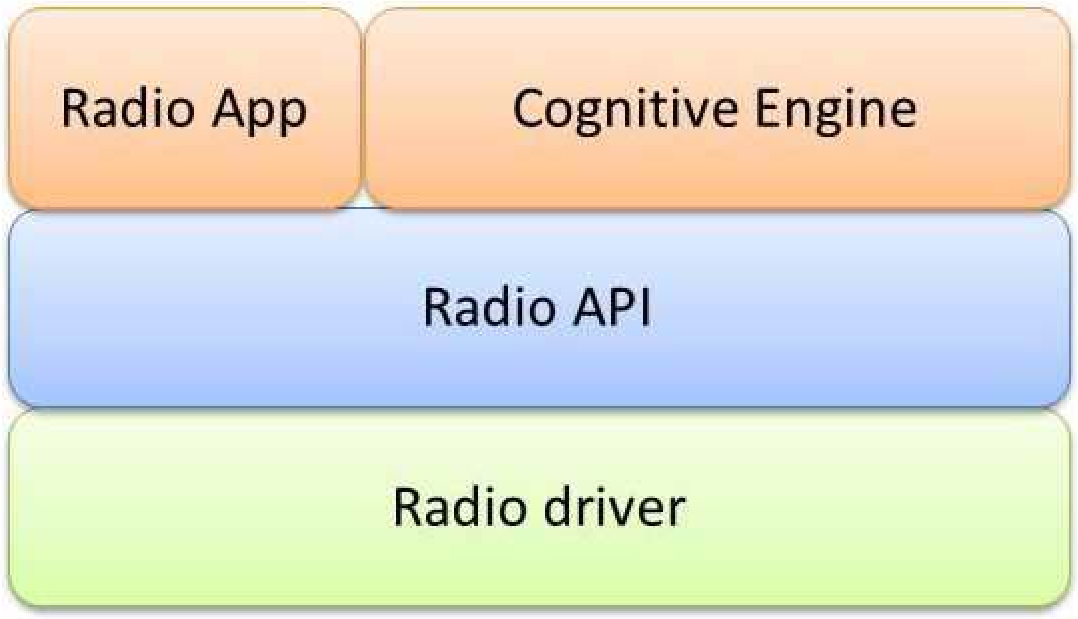


Figure 20. AVEP radio software stack.

An application is built on top of the API which is in turn built on top of the driver. Figure 21 shows how the radio software stack corresponds to SKIRL hardware. Applications and the API correspond to the GPP component in the hardware stack, while the driver crosses the divide between the RF module and the GPP component. The register access functions correspond directly to the RF module, while the drivers' helper functions correspond to the GPP component.

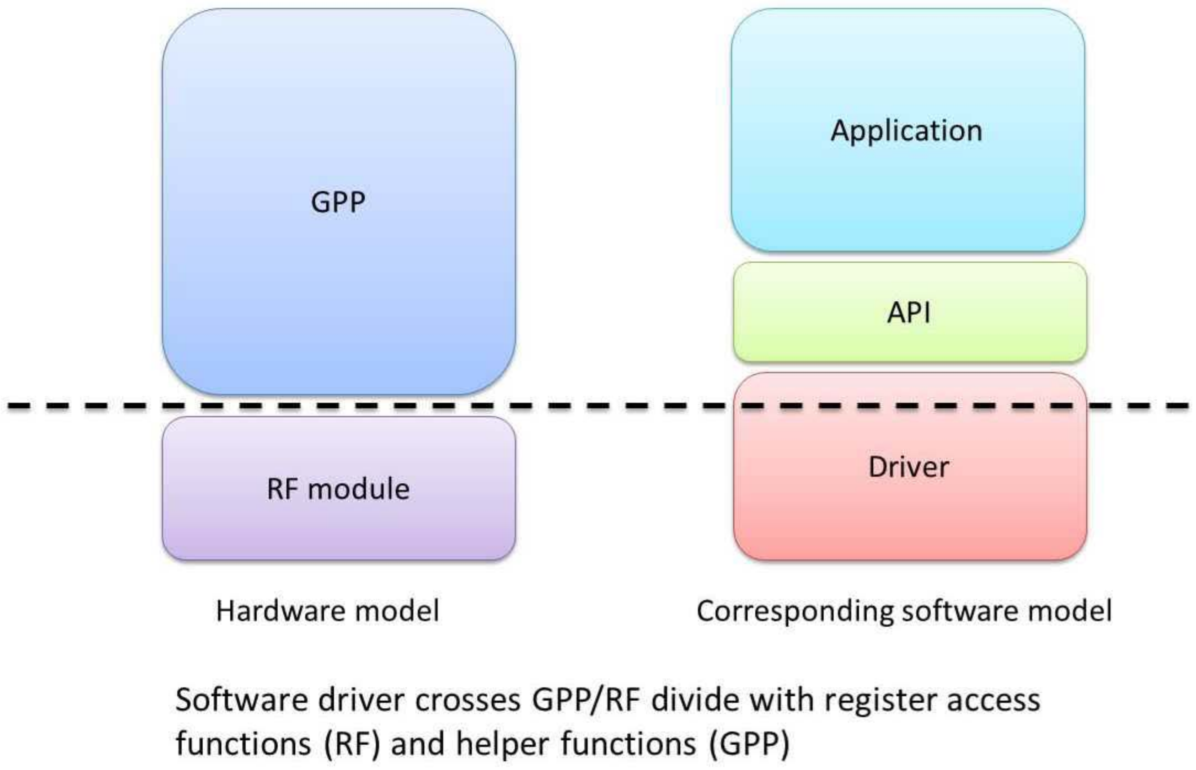

GPP
Application
API
RF module
Driver
Hardware model
Corresponding software model
Software driver crosses GPP/RF divide with register access functions (RF) and helper functions (GPP)

Figure 21. Radio software stack compared to SKIRL hardware components.

3.3.3.2.3.1 Radio Driver

The radio driver is the base interface for all interaction with the RF module. The radio drive uses the Python SPI driver to enable the bit-level communication with the RF module, and provides direct access to the RF modules configuration registers using register-specific functions. An example is:

```
_set_reg_operating_mode_1(0x01)
```

The publicly accessible functions are designed to be more user-friendly, using strings for input rather than hexadecimal values. This improves code readability and eases debugging. Many radio operations such as setting the frequency require reading and setting multiple registers, and the helper functions consolidate these multiple operations into a single function.

3.3.3.2.3.2 Radio API

The radio API is the interface for operating the radio module. It provides:

- Radio initialization,
- RF front end T/R switch control,
- Access to the air interface for listen, receive, and transmit operations,
- Timeout and random back-off, and
- Interrupt monitoring.

The radio API provides an initialization function that is responsible for setting up GPIO-based communication lines and configuring the radio module in a default mode. This allows a user to start using the radio module quickly with little effort. The default configuration enables radio operation at 434 MHz using GFSK at 4.8 kbps. Default payload length is 17 bytes. This default configuration is based on sample code provided by Hope RF [33].

The GPIO-based communication lines are used by the API's T/R switch to control the transmit and receive antennas. Each antenna is controlled by a single GPIO line, switching the antenna on or off. The T/R switch function has three states tx, rx, and off. Internally, the T/R switch always disables one antenna before enabling the other.

Hardware interrupts are used to indicate that certain events have occurred, including when a packet has been received and a packet has been sent. The system enables the specific interrupt for a particular event, and then loops to perform an action until the interrupt occurs. In this case, the interrupt port is tied to a GPIO line. When the radio module generates an interrupt, the GPIO line is driven low and the value GPIO line can be read by software.

A central service provided by the API is access to the air interface, using the listen, receive and transmit functions.

3.3.3.2.4 Motion Software

As with the radio software, the motion software operates on-board SKIRL, but is responsible for communicating with and controlling an off-board component, in this case the NXT brick. While the NXT brick is capable of running programs on its ARM processor, we have instead chosen to control the sensors directly from SKIRL, effectively using the brick as a pipe through which to control the sensors and rotors. We are using a Python library called nxt-python that provides direct access to the brick and its components over universal serial bus (USB). Unlike the radio software, there is no explicit motion driver. Where the radio driver provides basic connectivity to the RFIC, connectivity to the brick is provided by the nxt-python library.

3.3.3.2.4.1 Motion API

The motion API the core of the AVEP motion system, and is the interface for operating the motion module. It provides:

- Rotor and sensor initialization,
- Rotor state information,
- Motion actions such as “go forward”, “halt motion”, and “find line”.

The AVEP system uses a single connection to the NXT brick for all communication and control. This connection is established at the highest level, and passed to the motion API by the controller. The motion API uses the connection to initialize the AVEP rotors and sensors.

The simple “go forward” function is the basis of all AVEP forward motion. The left and right rotors are engaged and allowed to run until the “halt motion” function is called, which applies a braking function to the rotors. The “find line” function turns the AVEP in place about its vertical axis until a line is detected. The AVEP turns first clockwise then anti-clockwise, sweeping out increasing arcs until the light sensor detects a line beneath it. These three functions combine to provide a more sophisticated motion algorithm.

3.3.4 Control Software

This section presents the algorithms that provide the operational logic for the AVEP; the logic flow of the controller, the data collection algorithms used by the sensors, and the logic flows of the RF and MOT subsystems.

3.3.4.1 Controller

The controller is the central system component, and the hierarchical top layer of the system. The controller starts and manages the other modules, coordinating interactions between all the various subsystems.

3.3.4.1.1 Controller Finite State Machine

As the controller is the core of the AVEP system, so the FSM (shown in Figure 22) is the heart of the controller. The FSM consists of five states (“first_time”, “before_traverse”, “traverse_path”, “after_traverse”, “return_to_beginning”) and these five states handle all the functions of the AVEP after initialization. The FSM uses the “fsm_state” variable to determine the current state.

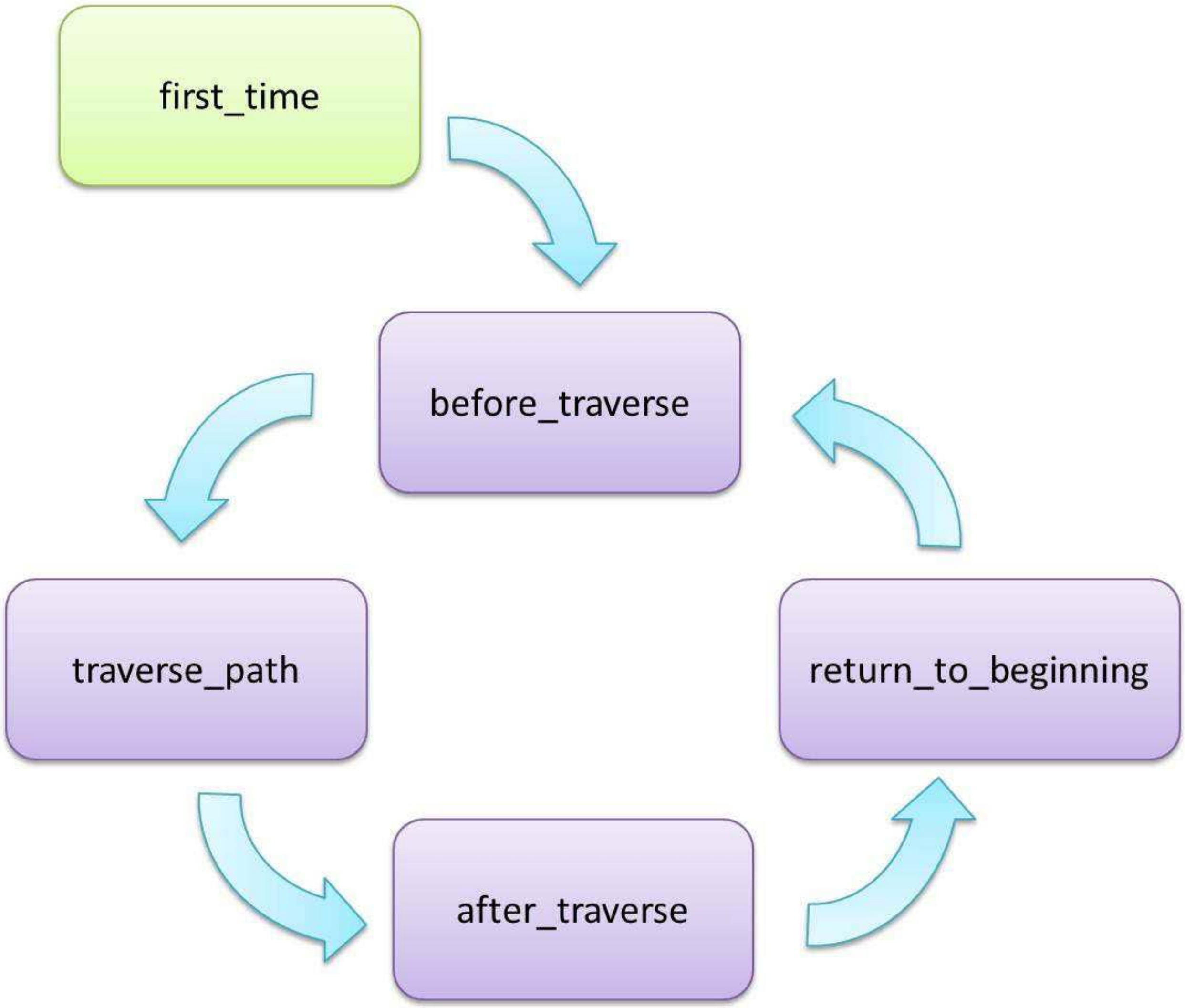


Figure 22. AVEP controller finite state machine.

3.3.4.1.2 Sensors
The AVEP uses three sensors for data collection: NXT light sensor, NXT color sensor, and barcode reader.

3.3.4.1.2.1 NXT Light Sensor
As mentioned previously, the NXT light sensor is used in the motion control algorithm. The light sensor is a NXT native sensor that is connected to the NXT brick, and the AVEP uses the nxt-python library to control it. The light sensor uses reflected light to determine the presence of the test track line beneath it. The threshold value for the light sensor for the test track line is 500. If the light sensor reads a value above the threshold, the test track line is not present beneath the light sensor, and if the sensor reads a value below the threshold, the test track line is present beneath the light sensor.

3.3.4.1.2.2 NXT Color Sensor
The NXT color sensor is used in the target tracking algorithm. As with the light sensor, the color sensor is a NXT native sensor that is connected to the NXT brick, and the AVEP uses the nxt-python library to control it. The color sensor uses reflected light to determine the color of an object below the sensor.

A different color is used to represent the targets and anti-targets; in this case, yellow represents a target, and red represents an anti-target. These values were chosen to minimize false positive cases. During a path traversal, every value recorded by the color sensor is stored in a single vector. After the path has been traversed, the results are processed in the following manner. The single vector is copied so that there are now two identical vectors, although one represents targets, and the other represents anti-targets. In the target vector, every vector element that is not equal to the target color value is set to 0. Likewise, in the anti-target vector, every vector element that is not equal to the anti-target color value is set to 0. Each vector is then processed to determine the number of transitions, that is, the number of times that an element is zero while the two following elements are non-zero. The number of transitions accurately indicates the number of targets (or anti-targets as appropriate) observed along the path just traversed.

3.3.4.1.2.3 Barcode Reader
The barcode sensor is used to determine the AVEPs position. We note that in an indoor scenario, a simple, cheap and accurate method of determining position is to reference an object with known location. This led to the idea of using barcodes at specific locations to fix a position, and using a barcode reader to determine the presence of a barcode, thus indicating current position [34].

As mentioned previously, despite being mounted on the chassis, the barcode reader connects directly to the SKIRL platform over USB. The barcode reader is accessible through the SKIRL OS device driver interface /dev. The specific interface for the barcode reader is /dev/hidraw0. The barcode reader itself operates in a mode that continually scans for barcodes, and once it scans one, the data embedded in the barcode is immediately available for decoding. Using the information available from [35], a barcode can be translated into a number, indicating a specific location on the test track.

The barcode reader is a threaded module, running in a continuous loop separate from (but started by) the controller. The value encoded in the barcode is relayed to the controller using a callback passed to the barcode reader when it is instantiated.

3.3.4.1.3 Radio Subsystem
The radio subsystem is responsible for all AVEP communications operation. The radio subsystem runs in its own thread, allowing it to run concurrently with, but independent of the main process. The radio

subsystem is originally instantiated by the AVEP controller, after which the radio subsystem FSM takes over.

3.3.4.1.3.1 Radio Subsystem Finite State Machine

The radio subsystem FSM is responsible for all AVEP RF operations. The AVEP controller uses the subsystem function set_state(current_state) to set the radio subsystem state of operation. The controller consists of five states: “stop”, “stream”, “update”, “reconfigure”, and “listen”. Each FSM state is responsible for a different aspect of RF operations.

The “stop” state is the default state of operation for the radio subsystem FSM. When the controller starts the radio subsystem, the FSM drops into this state, as it does when it completes any of its operations, represented by the other FSM states.

The “stream” state is used to stream data from the AVEP to a NBR. While the AVEP traverses a path, it transmits data packets to the NBR. The data payload inside the packet is not a significant aspect of this research, but payloads could include any type of operational data as required by the mission. (In our first round of AFRL funded research, payload data included video images captured by network nodes [118].)

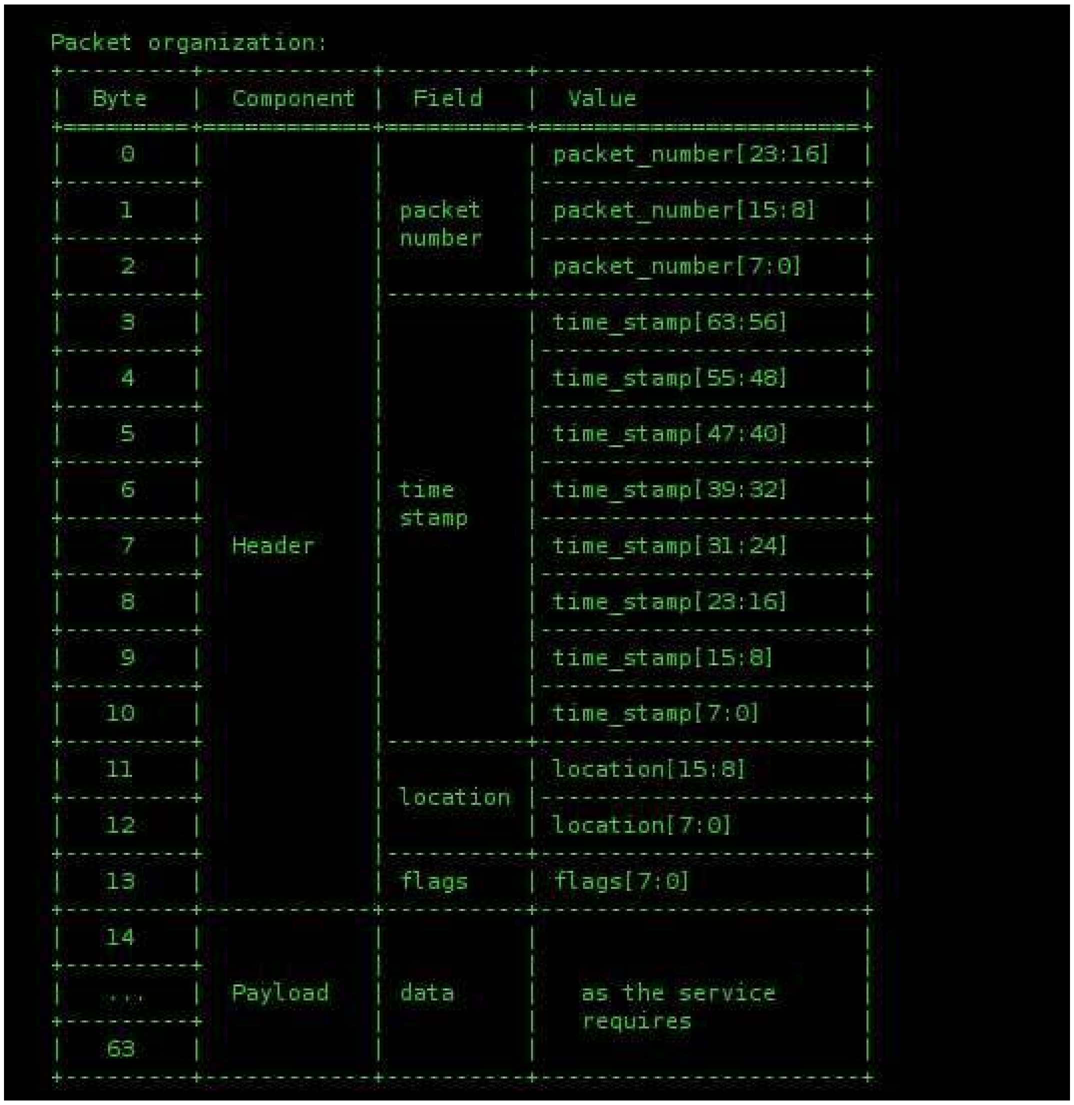
The “update” state is used by the AVEP to request updates from the NBR. Each time the AVEP completes a path traversal, it sends a request to the NBR for an update. When the NBR receives the request, it replies by transmitting a packet containing information on the number of packets that the NBR received from the AVEP while the AVEP was traversing its selected test bed path. This information can be used by the AVEP to improve its understanding of the RF environment.

The “reconfigure” state is used by the AVEP to notify the NBR of a new RF configuration. Every time the AVEP reaches Node 1 on the test bed, the AVEP makes a decision about how to operate. The solution generated by the decision maker at this point can include a new RF operational profile: frequency, modulation, bit rate, etc. However, If the AVEP unilaterally changes its operational profile, the receiving NBR won’t be able to receive the data, as it is still operating using an old profile. For this reason, using the original operational parameters, the AVEP notifies the NBR of the new operational profile, and waits for the NBR acknowledgment. When the AVEP receives the reconfiguration acknowledgment, it then reconfigures its own radio using the new parameters, knowing that the NBR has done the same, and that both RF systems are ready to start communicating using the new RF profile.

The “listen” state is used by the AVEP for RF sensing. In its initial iterations of operation, the AVEP explores its environment. During this exploration, the AVEP controller sets the state of the radio subsystem to “listen”, causing the radio subsystem to record RSSI along the path. This RSSI information is used by the decision making module to determine some RF path characteristics

3.3.4.1.3.2 Packet Structure

The RFM22B RFIC provides a very flexible platform for RF applications, but the downside to the flexibility is that it does not provide pre-existing support for packet structure or protocols above the PHY. To support AVEP operation, we developed a packet structure based on the transmission control protocol (TCP) standard [36]. The packet structure is shown in Figure 23.



```
Packet organization:
+----------+------------+----------+------------------------+
| Byte     | Component  | Field    |  Value                 |
+==========+============+==========+========================+
|    0     |            |          | packet_number[23:16]   |
+----------+            |          |------------------------+
|    1     |            | packet   | packet_number[15:8]    |
+----------+            | number   |------------------------+
|    2     |            |          | packet_number[7:0]     |
+----------+            |----------+------------------------+
|    3     |            |          | time_stamp[63:56]      |
+----------+            |          |------------------------+
|    4     |            |          | time_stamp[55:48]      |
+----------+            |          |------------------------+
|    5     |            |          | time_stamp[47:40]      |
+----------+            |          |------------------------+
|    6     |            | time     | time_stamp[39:32]      |
+----------+            | stamp    |------------------------+
|    7     | Header     |          | time_stamp[31:24]      |
+----------+            |          |------------------------+
|    8     |            |          | time_stamp[23:16]      |
+----------+            |          |------------------------+
|    9     |            |          | time_stamp[15:8]       |
+----------+            |          |------------------------+
|   10     |            |          | time_stamp[7:0]        |
+----------+            |----------+------------------------+
|   11     |            |          | location[15:8]         |
+----------+            | location |------------------------+
|   12     |            |          | location[7:0]          |
+----------+            |----------+------------------------+
|   13     |            | flags    | flags[7:0]             |
+----------+------------+----------+------------------------+
|   14     |            |          |                        |
+----------+            |          |                        |
|   ...    | Payload    | data     |   as the service       |
+----------+            |          |   requires             |
|   63     |            |          |                        |
+----------+------------+----------+------------------------+
```

Figure 23. RF subsystem packet structure.

Most of the header fields are self-explanatory. The “packet number” field contains a three byte integer indicating the number of the current packet. The “time stamp” contains an eight byte value indicating the time the packet header was packed, or put together. The “location” field contains a two byte integer that corresponds to the most recent location of the radio node. For the AVEP, this corresponds to the last barcode the AVEP recorded. The “flags” field is used to provide control information for coordination between transmitter and receiver nodes. Figure 24 shows the flags available to a Node A radio (NAR), such as the AVEP.

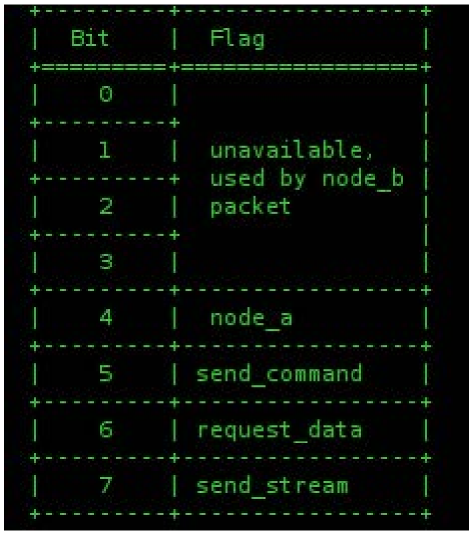

```
+---------+-----------------+
|  Bit    |  Flag           |
+=========+=================+
|    0    |                 |
+---------+                 |
|    1    |  unavailable,   |
+---------+  used by node_b |
|    2    |  packet         |
+---------+                 |
|    3    |                 |
+---------+-----------------+
|    4    |  node_a         |
+---------+-----------------+
|    5    | send_command    |
+---------+-----------------+
|    6    | request_data    |
+---------+-----------------+
|    7    | send_stream     |
+---------+-----------------+
```

Figure 24. Organization of flags field in RF subsystem packet. This figure shows the flags available to a Node A radio.

The “node_a” flag is used to indicate that the packet is coming from the NAR. The “send_command” is used to indicate to that the NAR is passing a reconfiguration command to the NBR. The “request_data” flag is used by the NAR to request an information update from the NBR. The “send_stream” flag is used to indicate that the NAR is send a stream of data to the NBR while the AVEP is traversing a path.

3.3.4.1.4 Motion Subsystem

Similar to the radio subsystem, the motion subsystem is responsible for all AVEP motion. The motion subsystem also runs in its own thread, again for concurrent but independent. The motion subsystem is originally instantiated by the AVEP controller, after which the motion subsystem FSM takes over.

3.3.4.1.4.1 Motion Subsystem Finite State Machine

The motion subsystem FSM is responsible for all AVEP motion operations. The AVEP controller uses the subsystem function “set_state(current_state)” to set the motion subsystem state of operation. The controller implements only two states: “stop”, and “go”. The “stop” state is the default state of operation for the motion subsystem FSM. When the controller starts the motion subsystem, the FSM drops into this state, as it does when it completes its actual motion operation, as represented by the “go” state.

The “go” state is used by the AVEP controller to initiate operational motion. This motion operation is governed by a motion behavior described below. The behavior is employed until the AVEP reaches the end of its chosen test bed path, indicated by arrival at test bed Node 2.

3.3.4.1.4.2 Motion Behavior: Follow The Line

The motion subsystem uses a line-following algorithm. This is a simplistic method of implementing robot motion, where the robot uses a sensor to detect a line on the ground, and the robot follows the as it moves forward [120]. The algorithm trades off between two sub-behaviors, “follow the line” and “find the line”. The “follow the line” state causes the AVEP to move forward in a straight line until the light sensor indicates that the AVEP is no longer following the line. At this point, forward motion is halted and the AVEP proceeds to “find the line”. The AVEP starts to turn in place about its z-axis, sweeping out larger and larger arcs as it seeks to find the test bed path—the line—with the light sensor. When the path has been found, turning motion is halted, and forward motion is again commenced via the “follow the line" behavior. Using the "follow the line" and "find the line" sub-behaviors, the AVEP is able to move effective along test bed paths by following the black lines marked out on the test bed.

3.3.4.2 Path Data Structure

CRs and AVs both require relevant and up-to-date information on their environment to operate effectively. AVs often use evidence grids to represent environmental data [3], while CRs commonly use internal or external databases to store RF information [37], [38]. We have developed a method inherently suited to multi-domain knowledge storage that uses graph structures to represent environmental data. The PDS is the system's central data storage component, labeled as "(Multi-Domain) World View" in Figure 4.16.

A single PDS is a software object that represents a graph edge, specifically a single test bed path. It can be instantiated multiple times to represent multiple paths. As shown in Figure 25, the test bed graph — originally presented in Figure 3—contains three individual paths between Node 1 and Node 2. Each path is a route that the AVEP can traverse as it moves from Node 1 to Node 2, and each path is represented by a single PDS.

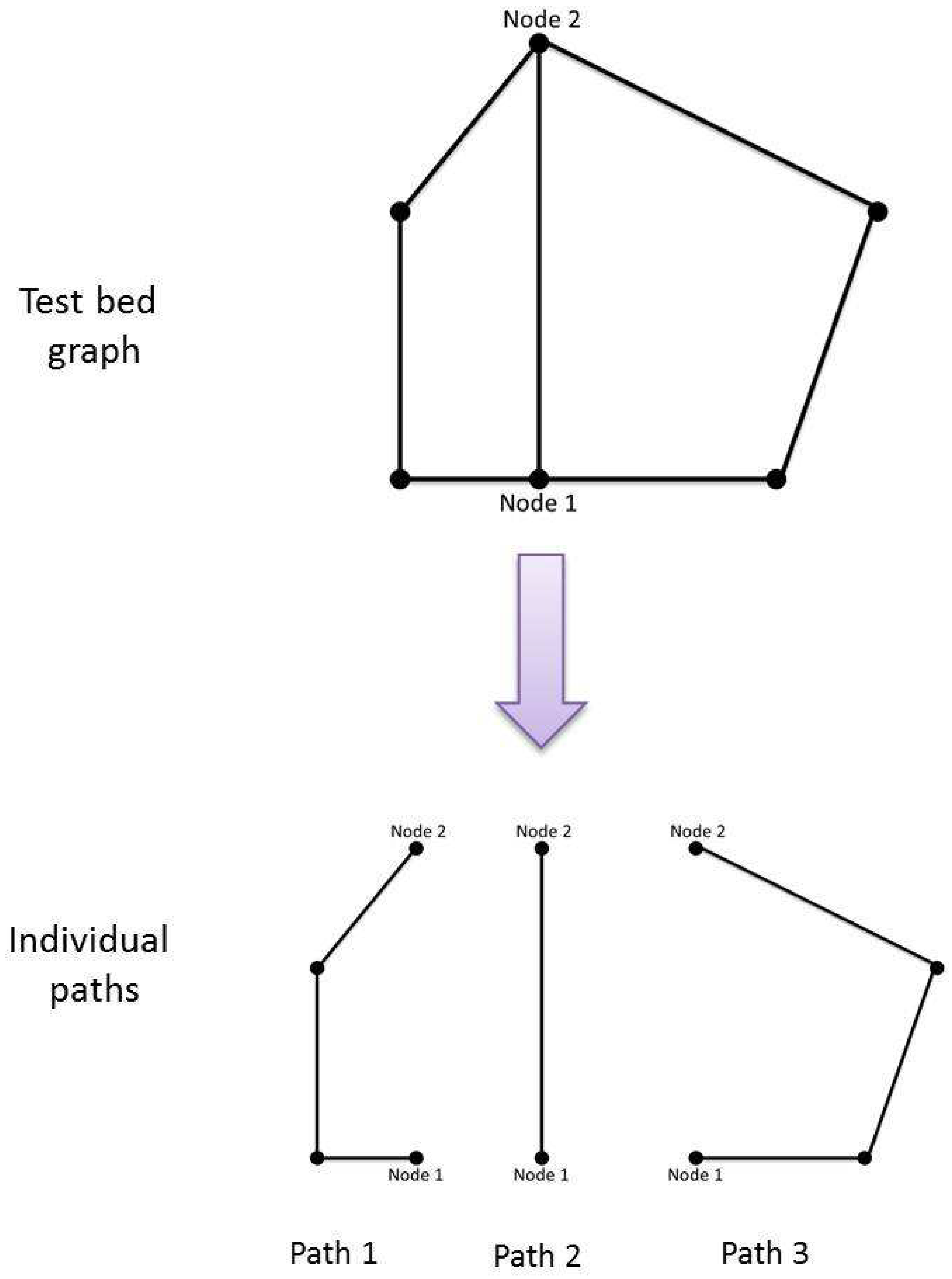


Figure 25. The test bed graph contains three separate paths between Node 1 and Node 2.

The PDS extends the concept of a weighted graph [39], or more specifically a weighted edge. Weighted graphs use weight to represent some cost associated with a particular graph edge, such as the distance between two cities in the traveling salesman problem, or the distance between two routers in a network. The PDS builds on the concept of an edge weight by using the weight as a data storage mechanism. The edge weight becomes multiple weights, each one representing a particular environmental characteristic, and all describing that particular graph edge.

For this research, each of the three instantiated PDS objects maintains physical and RF characteristics of a particular test bed path, such as path name, path distance or length, whether the path has been explored or not, and system knobs and meters along the path. Additionally, each PDS maintains a record of the most recent system solution for the path, as determined by the decision making module. Further solution details are presented in Section 6.3.6.2.

Currently the experimental test bed contains only three paths, each of which connects the same two nodes, Node 1 and Node 2. However, in a more complicated test bed layout, there would be additional nodes with additional edges and their attendant paths. An example is shown in Figure 26. In such a scenario, path search algorithms such as Dijkstra's algorithm [40] or D* Lite [41] can be used to determine a path of travel for the AVEP that spans multiple edges. The path search algorithm would use the data stored in each PDS object to evaluate each possible route.

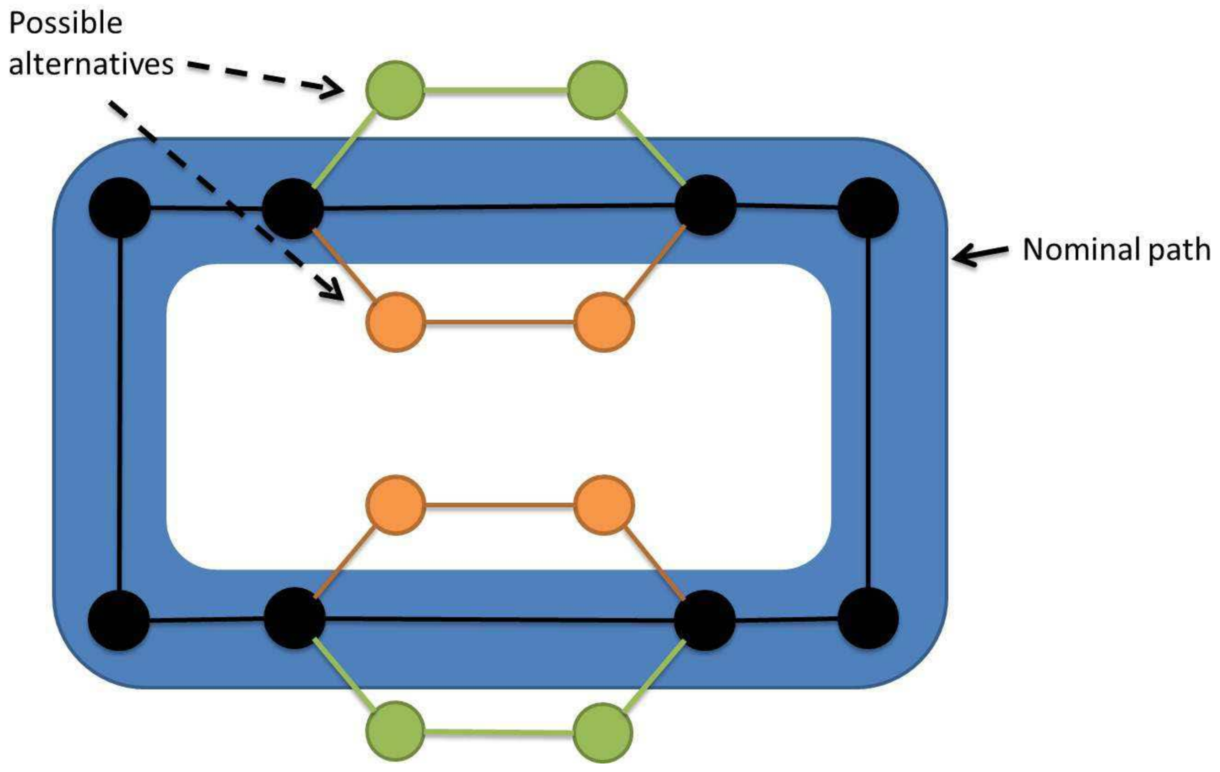
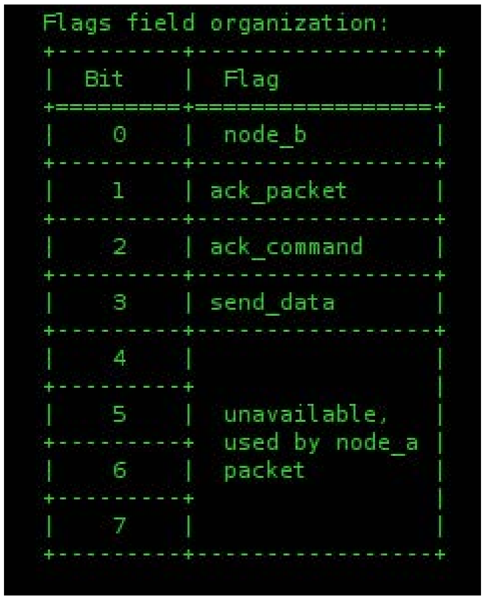


Figure 26. A more complicated test bed layout with additional nodes and edges, and their attendant paths.

3.3.5 Node B Details

The NBR is a standalone radio node based on the SKIRL radio platform. It is based on the same RF stack that provides radio functionality in the AVEP, but has no motion capabilities, in hardware or in software. The primary function of the NBR is to provide a node with which the AVEP can communicate during its operation.

The NBR uses a FSM to switch between three states: “listen”, “receive”, and “transmit”. The default state is “receive”, where it waits to receive a transmission from the AVEP. When it receives a packet from the AVEP, it parses the packet header, reading the value of the flags in the “flags” field, and sending an appropriate reply as necessary. Figure 27 shows the flags available to a NBR in replying to control flags from a NAR.



```
Flags field organization:
+---------+-----------------+
|  Bit    |  Flag           |
+=========+=================+
|    0    |  node_b         |
+---------+-----------------+
|    1    | ack_packet      |
+---------+-----------------+
|    2    | ack_command     |
+---------+-----------------+
|    3    | send_data       |
+---------+-----------------+
|    4    |                 |
+---------+                 |
|    5    |  unavailable,   |
+---------+  used by node_a |
|    6    |  packet         |
+---------+                 |
|    7    |                 |
+---------+-----------------+
```

Figure 27. Organization of flags field in RF subsystem packet. This figure shows the flags available to a Node B radio for communication with a Node A radio.

If an incoming packet contains the “send_stream” flag, the NBR records the packet number of the incoming packet, and then returns to “receive” state to await another packet. If an incoming packet contains the “request_data” flag, the NBR calculates the total number of data stream packets it has received since the last update. Before it transmits this information back to the NAR, it enters the “listen” state to implement LBT as described in previous reports. When the channel is clear, the NAR enters the “transmit” state and transmits the data back to the NAR, using the “send_data” flag, and returns to the “receive” state. If an incoming packet contains the “send_command” flag, the NBR parses the packet to determine the new radio configuration to implement. It then enters “listen” before transmitting an acknowledgment to the NAR using the using the “ack command” flag. The NBR then reconfigures its radio parameters and once again enters the “receive” state to await communications from the NAR.

3.3.6 Learning and Decision Making Algorithms

The learning and decision making processes make up the cognitive core of this research, providing intelligent action and introspection for the AVEP. Learning and decision making in the AVEP are closely entwined; learning feeds decision making, and decision making feeds learning. The learning and decision making processes presented here are the essence of UMDDM.

Decision making "is the process of selecting a possible course of action from all the available alternatives," [42]. Sometimes this is also referred to as reasoning [43]. Clearly the ultimate point of making a decision—selecting a possible course of action—is actually to act. Decision making is the process of deciding on a course of action, which is passed to the RF and MOT subsystems, the system's actuators.

The *Oxford English Dictionary* defines the verb "learn" as "to acquire knowledge" specifically "as a result of study, experience, or teaching" [44]. However, the concept of what is machine learning is a moving target. Machine learning is closely tied to artificial intelligence (AI), and therefore subject to the "odd paradox", the concept that once AI has solved a problem, that problem and the corresponding solution no longer belong to the domain of AI [45], [46]. Karen Haigh differentiates between adaptation, where a system can change its behavior based on current conditions; and learning, wherein a system uses its experience to change its adaptation methods, effectively adaptive adaptation [46]. In this research, learning is used in two distinct ways, learning the environment, and learning from experience. Learning the environment provides information for decision making, while learning from experience evaluates the decisions and provides feedback to the decision maker.

The AVEP cognitive process implements a cycle similar to Mitola's cognition cycle [47], and the observe, decide, and act (ODA) loop [48], and bringing to bear the observation noted in Section 5.2, namely that CRs and AVs perform similar tasks, albeit in different domains:

- Analyze their environment,
- Make and execute a decision,
- Evaluate the result (learn from experience), and
- Repeat as required.

The ODA loop and the observations above are so similar that they can be effectively combined into a single loop, shown in Figure 28. The specific cognition process the AVEP uses is shown in Figure 29.

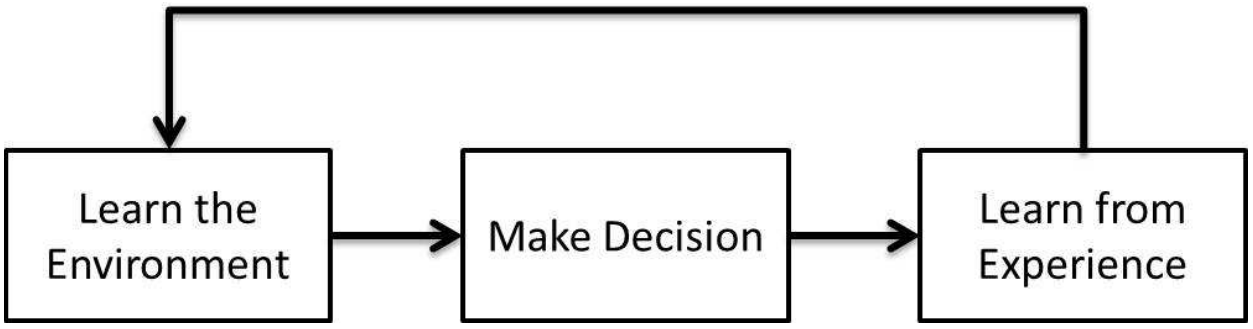


Figure 28. The ODA loop applied to CR and AV scenarios.

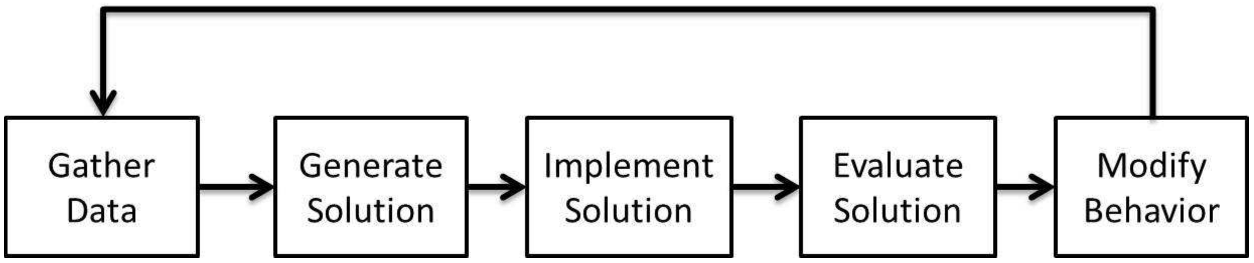


Figure 29. The cognition cycle used by the AVEP.

3.3.6.1 Learn The Environment

There is a significant body of research that deals with decision making and learning, from perspectives that include pure AI, economics, child development, CR, and AVs. Much of what is published deals with the decision making and learning processes themselves, skipping right over the acquisition of information that supports these processes. The AVEP learn the environment process is responsible for gathering this information, collecting meter values during AVEP operation.

The AVEP uses its sensor systems to learn the environment. The AVEP is programmed with some initial information to jump-start the learning process. This includes path length for individual paths in the test bed. In the AFRL systems that this research supports, UAVs fly preprogrammed flight paths, and the AVEP emulates this setup with a priori knowledge of the test bed paths. Information such as number of targets and anti-targets along the path as well as RF noise are sensed and stored as the AVEP actually travels the test bed paths. The mechanics of the environmental learning process are discussed in Section 6.3.4.1.1, while the individual sensors used to gather the information are presented in Section 6.3.4.1.2. It should be noted that while the AVEP gathers its initial information from during an exploration phase, and then moves into an “exploitation” phase (using the information it has obtained to carry out its operational duties), the AVEP continues to gather environmental information about its current path of travel during operation. Additionally, the AVEP maintains a record of the last time it received information about a given path. If the meters corresponding to a given path have not been updated within a certain number of iterations, the AVEP switches from operational (or “exploit”) mode to exploration mode, and explores that path to updates its internal path data. In this manner, the AVEP is able to maintain up-to-date information about its changing environment, ensuring effective decision making with relevant information.

3.3.6.2 Decision Making

Decision making is the process of choosing an action or outcome from a set of possible actions or outcomes. Optimization theory often refers to multiple criteria decision making (MCDM), while CR research often uses multi-objective optimization (MOO), but they both refer to decision and planning involving multiple conflicting criteria that should be considered simultaneously [49]. Rondeau mentions several methods for evaluating multi-objective optimization problems, including the weighted-sum approach, the linear-logarithmic function, and the constant-elasticity-of-substitution function [5]. Rondeau ultimately uses evolutionary based genetic algorithms (GAs) to address his MOO problems, and this approach has been embraced by researchers in a wide variety of application domains, including economics [50], mechanical engineering [51], and cryptography [52]. Guided search algorithms like GAs are well suited to finding solutions in large search spaces. Additional techniques that have been investigated for guided search include simulated annealing [43] and swarming algorithms [53].

Zitzler and Theile present a general formulation for multiple objective optimization problems in [54], shown in (1).

$$\begin{aligned} min/\max \bar{y} &= f(\bar{x}) = f_1(\bar{x}), \dots, f_m(\bar{x})) \quad (1) \\ subject\ to\ \bar{x} &= (x_1, x_2, \dots, x_m) \in X \\ \bar{y} &= (y_1, y_2, \dots, y_n) \in Y \end{aligned}$$

That is, we seek to minimize (or maximize) a vector function f that maps a tuple of m input parameters to a tuple of n output parameters, or objectives. The set $\bar{x}$ is the set of input parameters, and $\bar{y}$ is the of objective values determined by the objective functions. The set of solutions to a multi-objective problem lie on the Pareto front, which consists of all the solution sets $\bar{y}$ which are non-dominated, that is, those sets that cannot be improved in some dimension without a decrease in some other dimension. Multi-objective optimization problems are naturally problems in balancing trade-offs [55]. Attempting to minimize both BER and equivalent isotropically radiated power (EIRP) in the RF domain is a perfect example. With all other factors held constant, minimizing BER requires increased transmit power, while reducing transmit power correspondingly drives an increase in BER. In the MOT domain, for a given travel distance, attempting to minimize both travel time and vehicle velocity results in the same interplay. Multi-objective optimization balances the trade-offs, and provides decision making capability in the face of multiple competing decision criteria.

3.3.6.2.1 Objective Functions

This work is the first to combine flexibility in the RF with flexibility in the MOT domain. To implement this flexibility, any decision making process must take into account both domains in the objective functions used. In order to clearly show the concept of UMDDM, the proof of concept prototype AVEP implements only a few objective functions in total. We have chosen to follow the example set by DARPA in the DRC. Decision making will be based on a variety of factors, including mission success and completion time. For the RF domain, we have chosen to BER as one objective. BER is a commonly used metric for evaluating wireless communication systems. We also chose packet delivery to evaluate RF performance. Packet delivery is based on the concept of goodput or throughput, and integrates consideration of the chosen MOT metric. The single chosen MOT objective function is time, the time it takes the AVEP to traverse a single test bed path. In addition to the RF and MOT objective functions, the AVEP also uses a target/anti-target objective function to evaluate mission based parameters. The objective functions are explored in greater detail below.

This section presents the four objective functions used in the AVEP decision making process. For each objective, we list the required knobs, meters, and other objective functions used in the objective function calculation.

3.3.6.2.1.1 Target/Anti-target Score

Dependencies:

- Knobs: None
- Meters: Targets, anti-targets
- Objectives: None

We developed the target/anti-target score to incorporate consideration of mission success into the decision making process. While this work is modeled on AFRL UAV scenarios, we am not privy to the missions the United States Air Force (USAF) is flying. And although it likely goes without saying, we will explicitly state that we do not have access to USAF or AFRL UAV mission parameters. As a result, we developed the details of this objective function to model a plausible UAV mission objective, namely tracking and observing some target while avoiding some other anti-target. Further, we wished to show a reasonable tradeoff between the desirable aspects of the mission (finding targets) and the undesirable aspects of the mission (encountering anti-targets). Clearly, if the USAF uses a mission objective like this, the weightings would change based on the specific mission. The risk inherent in encountering a large number of anti-targets may be considered acceptable in order to complete a high value mission.

Targets (X) and anti-targets (Y) are mission-based parameters. They attempt to model mission priorities. Targets are objects that the AVEP should track, while anti-targets are objects that the AVEP should avoid, in the course of its mission. During AVEP operation, targets and anti-targets are represented by colored pieces of cardboard placed along the test bed paths. As the AVEP travels the path, it records the presence of the targets and anti- targets as described in Section 6.3.4.1.2.2. The target/anti-target score (Z) is an objective that is used to incorporate mission information into the decision making process. In the scenario presented in Section 1.4 (second round), a UAV carrying out a mission might reasonably be required to find and track a number of targets, while avoiding or minimizing detection by hostile entities (anti-targets). The Z function (2) defines a series of cases for possible values of X and Y, with instances where X > Y given greater value. Note that where $X \leq Y$, $Z \leftarrow 0$. This incorporates the mission directive to avoid anti-targets. A graphical representation of the objective function is shown in Figure 30, where $Z \sim f(X,Y)$ and $X \in \mathbb{Z}$, $X = \{x | 0 \leq x \leq 20\}$, $Y \in \mathbb{Z}$, $Y = \{y | 0 \leq y \leq 20\}$.

$$Z = \begin{cases} 0 & \text{if } X = 0 \text{ or } X < Y, \\ 0.2 & \text{if } X = 1 \text{ and } Y = 0, \\ 0.2*(X - Y) & \text{if } 1 < X <= 3 \text{ and } Y <= (X-2), \\ 0.15*(X - Y) & \text{if } 4 < X <= 6 \text{ and } Y <= (X-3), \\ 0.2*(X - Y) & \text{if } 4 < X <= 6 \text{ and } Y <= (X-2), \\ 0.2*(X - Y) & \text{if } X > 6,\ Y <= (X-4), \text{ and } 0.2*(X-Y) <= 1.0, \\ 1.0 & \text{if } X > 6 \text{ and, } Y <= (X-4), \text{ and } 0.2*(X-Y) > 1.0, \\ 0.15*(X - Y) & \text{if } X > 6 \text{ and } Y <= (X-3), \\ 0.1*(X - Y) & \text{if } X > 6 \text{ and } Y <= (X-2), \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

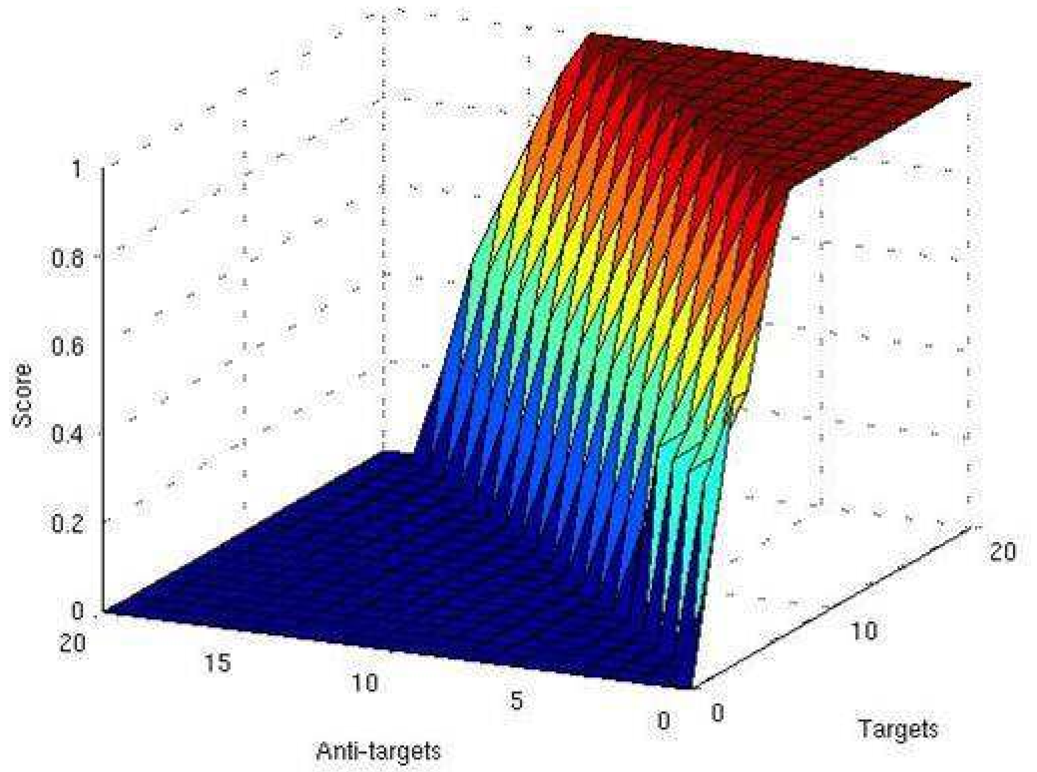


Figure 30. Graphical representation of Z objective function as a function of targets and anti-targets.

3.3.6.2.1.2 Time

Dependencies:

- Knobs: Rotor power
- Meters: None
- Objectives: None

Time refers to the length of time required for the AVEP to traverse a given path. The time required to travel a given distance at constant velocity is given by a standard first semester physics equation (3):

$$T = d/v \tag{3}$$

where T is time, d is the distance traveled, and v is the velocity. However in this case, while the distance traveled along a given path is known, AVEP velocity is not known. The available system knob is rotor power, and rotor power is controlled through a unitless number $\{P_{rotor}|64 \leq P_{rotor} \leq 128\}$. Without further exploration, there is no indication how this value relates to velocity. We ran repeated tests of the AVEP, driving it along a fixed length (0.762 meter) path using various rotor power values. Table 2 shows the values gathered during these experiments.

Table 2. Rotor power and time measurements for AVEP, used to determine AVEP velocity.

Rotor power	Time (sec)				
25	6.636803	7.384832	6.820400	7.033324	6.615562
35	6.011670	5.473605	5.078091	4.992305	5.963818
45	4.268262	4.179882	3.975959	4.217389	4.037789
55	4.829124	4.848656	3.488006	3.301235	4.727865
65	4.387924	5.452579	6.348041	6.948240	4.474748
75	5.597478	4.591388	6.627983	6.298144	7.116761

We generated a 3rd-order polynomial equation to fit the data, and a plot of the experimental data and a 3rd-order polynomial are shown in Figure 31. Using the now generated polynomial, we can determine the expected time to traverse a given distance with using a particular value of rotor power as input.

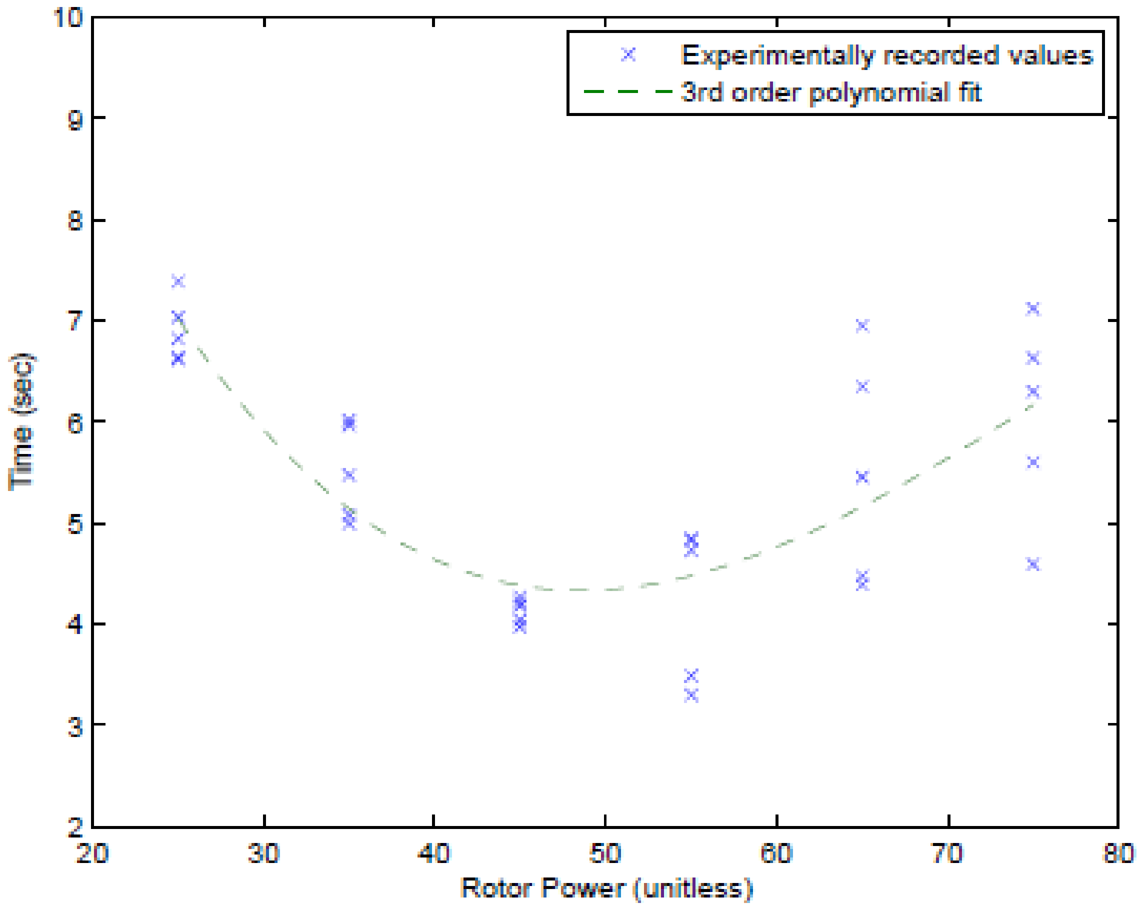


Figure 31. Experimentally recorded values of time (to travel a fixed-length path) and requisite AVEP rotor power values, with 3rd order polynomial fit.

While the experimental data and plotted data are valid for a path distance of 0.762 meters (30 inches), the calculation of the polynomial used in the calculation of objective function includes some additional steps. Knowing that the distance in the experiment is 0.762 meters, we divide the experimentally generated time results by the distance to come up with a unit time value, the time it takes the AVEP to travel a meter using the rotor power value currently under test (4).

$$\hat{t} = t_{exp}/d \qquad (4)$$

Multiple mathematical calculation packages provide the functionality for generating polynomial fits to data. The numerical Python package NumPy [56] provides the polyfit and poly1d functions for fitting data with polynomials. Using the polynomial generated, a simple function can provide the expected time the AVEP will require to travel a given distance with specified rotor power.

Note that as rotor power increases, the corresponding time first decreases, then starts to increase again. This is an artifact of the AVEP line-following motion algorithm. High rotor power causes high AVEP velocity for a very short period of time, but this velocity immediately goes to 0 when the AVEP moves off the line it follows. The AVEP must stop and reacquire the line, before starting to move forward again. And because the AVEP is moving faster when it diverges from its path, it travels farther from the path before stopping, and takes longer to reacquire the line.

Figure 32 shows a graphical representation of the objective function, where $T \sim f(D, P_{rotor})$ and $D = \{1.575, 1.219, 2.223\}$ (m), $P_{rotor} \in \mathbb{Z}$, $P_{rotor} = \{p | 5 \leq p \leq 80\}$.

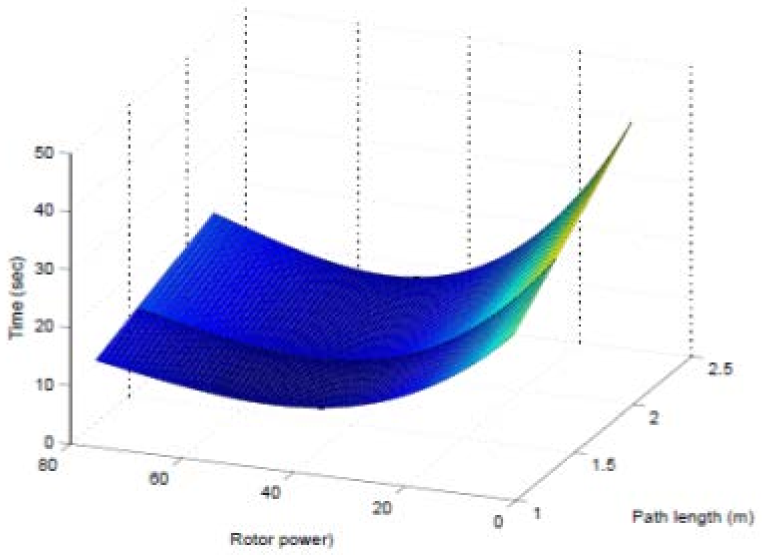


Figure 32. Graphical representation of T objective function as a function of path distance and rotor power.

3.3.6.2.1.3 Bit Error Rate

Dependencies:

- Knobs: Bit rate, EIRP
- Meters: RSSI
- Objectives: None

BER is a metric that is commonly used to evaluate the performance of communication systems. Equations for calculating BER vary according to the type of modulation employed, as well as the type of channel in use. For the purpose of this research, we have assumed an additive white gaussian noise (AWGN). The type of modulation employed is dictated by our choice of the RFM22B RFIC. While the RFIC can use FSK, GFSK, or OOK modulations, the AVEP currently only uses FSK. The RFIC data sheet does not provide many details about the inner workings of the RF modem, therefore in the absence of additional information, we have assumed that based on the low cost of the RFIC, it employs noncoherent detection. Sklar [57] provides an equation for calculating BER for noncoherently detected FSK (5):

$$B = \left(\frac{1}{2}\right) e^{-\frac{1}{2}\frac{E_b}{N_0}} \tag{5}$$

where B is the BER, and E_b/N_0 is the energy per bit to noise power spectral density ratio. While E_b/N_0 is commonly used in textbooks and theoretical research, it is a difficult concept to use in practical applications and implementations where SNR is easily and readily determined. Fortunately Sklar [57] also presents the relationship between E_b/N_0 and SNR for binary signals (6):

$$\frac{E_b}{N_0} = \frac{s}{n}\left(\frac{Bw}{R_s}\right) \tag{6}$$

where s/n is the (linear, non-dB) signal to noise ratio, Bw is the signal bandwidth, and R_s is the bit rate.

Due to the non-linear nature of FSK, analysis of FSK spectrum is non-trivial [58], however one source [59] provides (7) for calculating the bandwidth of binary FSK:

$$Bw = 2\left(f_{dev} + \frac{R_s}{2}\right) \tag{7}$$

where f_{dev} is the maximum frequency deviation, which can be read from a register on the RFIC. Through repeated measurements on spectrum analyzer, we found that the bandwidth can be approximated as Bw = 100 kHz across the full range of RFIC data rates.

A link power budget is used to calculate the received power at a receiving terminal (8):

$$P_R = P_T + G_{antenna} - L_p - L_{misc} \tag{8}$$

where P_R is the power level at the receiver, P_T is the power level at the transmitter or EIRP, $G_{antenna}$ is the antenna gain, L_p is the path loss, and L_{misc} represents miscellaneous losses not accounted for elsewhere. In this work, we have assumed that $G_{antenna} = 0$ and $L_{misc} = 0$. The modified link budget is shown in (9).

$$P_R = P_T - L_p \tag{9}$$

Pratt, Bostian, and Allnutt present a detailed discussion of path loss in [60]. Path loss is dependent on both frequency and distance. It would be difficult to calculate path loss with any degree of accuracy without knowing if mulitpath or shadowing are present. However, we can empirically determine a value for path loss in the test bed experiments through repeated measurements. We determined that the path loss experienced by the AVEP is Lp =90 dB.

A noise power budget can be used to calculate noise power; however we have approximated the noise power using RSSI measurements. RSSI can be read from a register on the RFIC. The RFIC data sheet provides a figure that graphs the relationship between input power in dBm and RSSI [20]. Using the table, we derived a simple linear equation for input power and RSSI (10).

$$P_{in} = \left(\frac{RSSI - 125.0}{2.0}\right) - 60.0 \tag{10}$$

Using (10), and reading RSSI values in the absence of any input signal, we have approximated the noise power as N = −92 dBm.

Figure 33 shows the standard BER vs. E_b/N_0 curve for noncoherent FSK, called a waterfall curve. Figure 34 shows the resultant E_b/N_0 values as a function of SNR in dB and R_s in kbps.

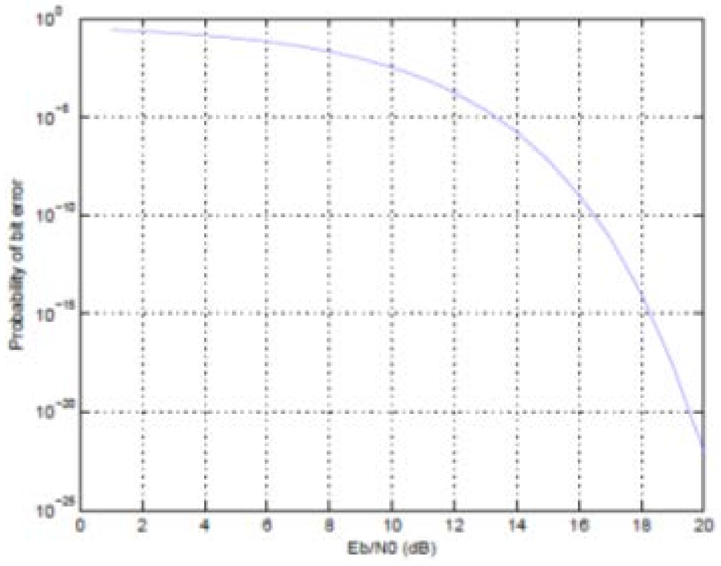


Figure 33. BER waterfall curve for noncoherent FSK.

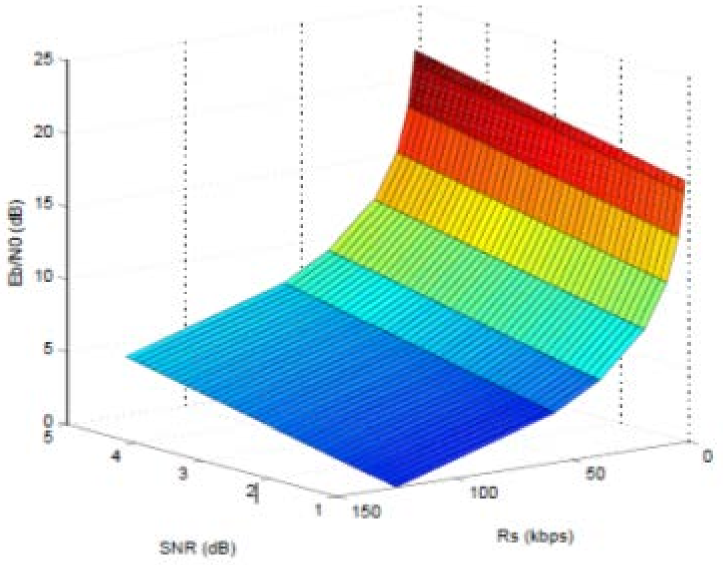


Figure 34. E_b/N_0 values as a function of SNR and R_s.

Combining Figure 33 and Figure 34 into a single plot yields Figure 35 where

$B \sim f(SNR, R_s)$ and $SNR \in \mathbf{R}$, $SNR = \{x | 1 \leq x \leq 15\}$, $R_s = \{2.4, 4.8, 9.6, 19.2, 38.4, 57.6, 125\}$ (kbps).

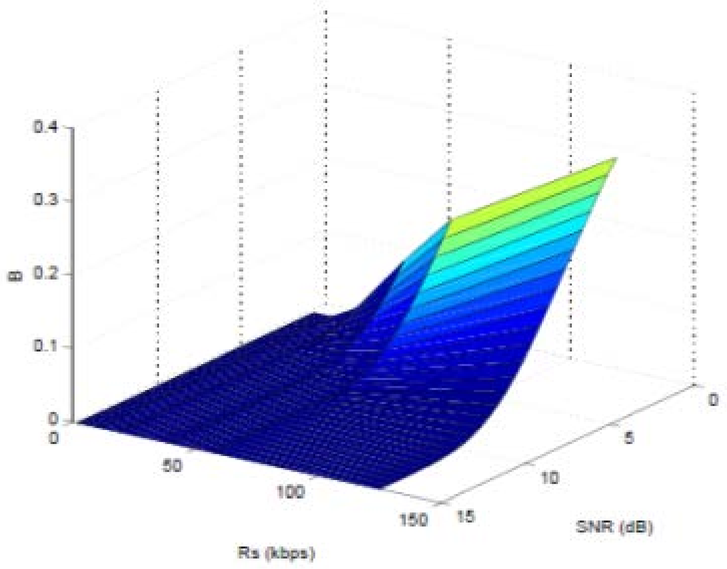


Figure 35. Graphical representation of B objective function as a function of SNR and R_s.

3.3.6.2.1.4 Packet Delivery

Dependencies:

- Knobs: Bit rate
- Meters: None
- Objectives: Time

Packet delivery measures the number of packets that can be transmitted by the AVEP while it traverses a single test bed path. The time it takes for a single packet to be transmitted and arrive at the other end of a link is given by (11):

$$t_{packet} = \frac{d_{prop}}{c} + \frac{l_{pkt}}{R_s} \tag{11}$$

where d_{prop} is the propagation distance, c is the speed of light, l_{pkt} is the length of the transmitted packet (number of bits) and R_s is the bit rate. The term d_{prop}/c represents the propagation delay, or the time it takes a signal to travel from one point to another through the air. The term $l_{pkt}\ /R_s$ represents the transmission delay, or the amount of time it takes to transfer all the bits of a packet on to the medium. Leon-Garcia and Widjaja refer to this as block transmission time [61].

In the case of repeated transmissions, system latency must be considered. System latency is the delay that occurs before the system can send another packet after it has completed transmitting a packet. System latency arises when the system must handle other tasks before it can resume sending packets. In the case of the AVEP, the system is managing sensor information as well as the MOT subsystem in addition to the RF subsystem, so system latency is nontrivial. However, we have defined packet delivery as the number of packets that can be transmitted by the AVEP while it traverses a single test bed path. This definition allows us to remove consideration for receiving the packet at the other end from the calculation. In fact, the time required to transmit packets is simply a function of the transmitter: how fast it can move the bits on to the medium, and how long it takes before it can repeat the process.

Therefore, the total number of packets the AVEP can transmit during a single path traverse is equal to the time it takes to traverse the path divide by the time it takes the AVEP to transmit the packets (12):

$$G = \frac{T}{t_{sys} + \frac{l_{pkt}}{R_s}} \tag{12}$$

where G is packet delivery, T is time, as calculated in Section 6.3.6.2.1.2., and t_{sys} is the system latency.

Packet delivery is similar to throughput or goodput, common metrics used to evaluate wireless network performance. While BER is a measure of RF performance in a particular channel, packet delivery formulated in this manner implicitly ties together MOT and RF parameters.

Figure 36 shows a graphical representation of the G objective function, where $G \sim f(T, R_s)$ and $T \in \mathbf{R}$, $T = \{t | 0 \leq t \leq 50\}$, $R_s = \{2.4, 4.8, 9.6, 19.2, 38.4, 57.6, 125\}$ (kbps).

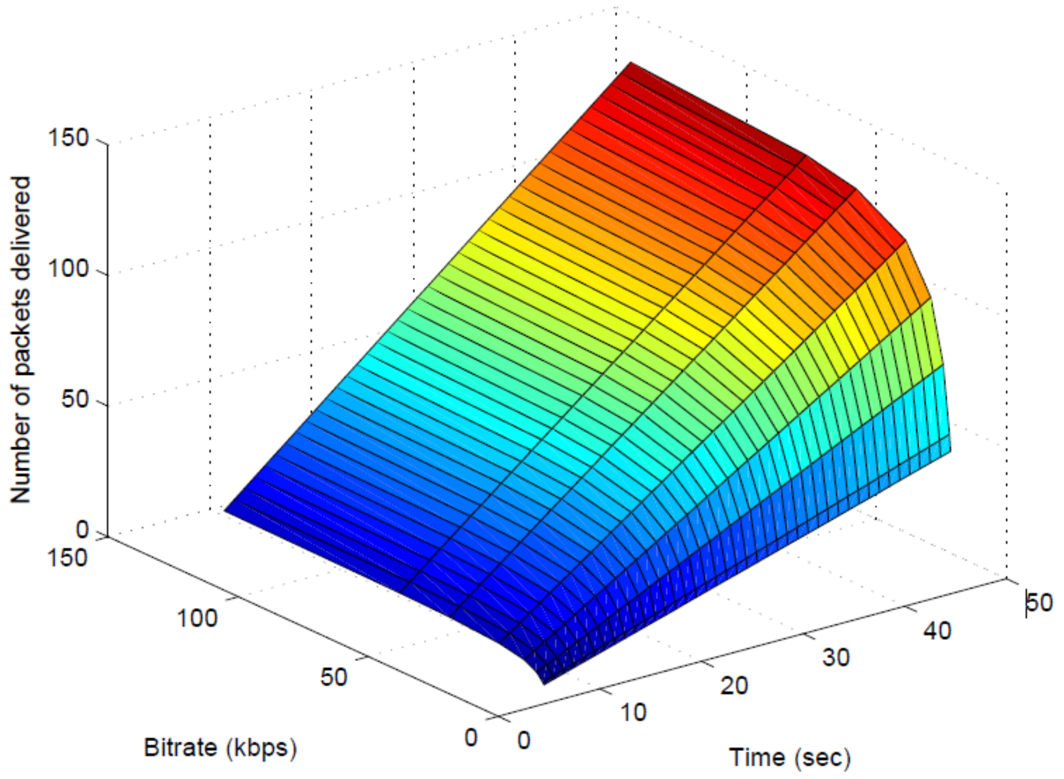


Figure 36. Graphical representation of G objective function as a function of time and bit rate.

3.3.6.2.2 Decision Making Process

The decision making process starts off with generating a solution space. This solution space contains all the possible solutions the AVEP can implement at a given moment. The decision maker module uses the system knobs and meters to calculate the objectives according to the equations in Section 6.3.6.2.1.2. A complete solution space is a list of solutions, where each element in the list is itself a list with objective values as elements (13),

$$S = \begin{matrix} [[Z_1, & T_1, & B_1, & G_1], \\ [Z_2, & T_2, & B_2, & G_2], \\ \vdots & \vdots & \vdots & \vdots \\ [Z_n, & T_n, & B_n, & G_n]] \end{matrix} \quad (13)$$

where S is the solution space, and $S_i = [Z_i,\ T_i,\ B_i,\ G_i]$ is an individual solution. The solution space is generated by looping through the input parameters, including the knobs and meters, and calculating each objective value in turn. This is possible because the solution space is rather small in this case, only 1152 solutions. At the same time that the decision maker generates the solution space, it also creates a corresponding parameter space *param*, where *param*$_i$ is a Python dictionary containing the input parameters that result in solution S_i.

The decision maker uses a nondominated sort to generate a population of suitable solutions. A vector **x** dominates vector **y** if no value of **x** is less than **y** and one value of **x** is strictly greater than **y**. In a collection of vectors, any one vector that is not dominated by any other vector in the collection is nondominated. The optimal solutions to a MOO problem are the nondominated solutions, also known as the Pareto-optimal solutions [62]. A nondominated sort is a tool used in MOO to determine which members of a population best satisfy a MOO problem. A large number of MOO problems are not expressed in terms of a common metric. Rapoport discusses the difficulty of dealing with multifaceted problems even when the problem can be expressed in terms of a single parameter, let alone problems that cannot be boiled down to a common metric [55]. Nondominated sort is one effective way for dealing with exactly these types of complex problems, by ranking populations according to how the components of individual members compare to the same components of other members in the population. The nondominated sorting method is used in the nondominated sorting genetic algorithm (NSGA) presented by Srinivas and Deb in [62]. As noted previously, the decision making solution space is relatively small, and a guided search method such as a GA is not required for effective search. Shi, Chen, and Shi isolated the nondominated sort algorithm in the NSGA [63], and we have used it to do a population based multi-objective sort on the search space.

The nondominated sort algorithm generates a list of nondominated results, as shown in [62], [63], from which a single solution is randomly selected for implementation by the AVEP. However nondominated does not necessarily mean “good”. Controlled experiments have shown that, given the right objective function inputs, it is possible that one or more nondominated solution exist where Z = 0 or B = 0.5. Clearly, these are not “good” values. And yet, in certain cases, these values show up in the nondominated results. This is where system policy comes into play.

Policy is often used to implement legal or procedural limits on CR operation [64],[8]. In this case, policy is used to indicate minimum and maximum desired values for individual solution objectives. The policy file contains the following lines:

```
min_z = 0.1

max_ber = 0.25
```

notifying the decision maker it is to select solutions that have a minimum Z value of 0.1, and a maximum B value of 0.25. When the decision maker selects a value at random from the current Pareto front, it checks the solution against the policy. If an objective value does not satisfy the policy, the decision maker chooses another solution and repeats the check. The first nondominated solution that satisfies the policy is the solution that the AVEP will implement. The decision maker returns the solution and the corresponding parameters to the AVEP controller for implementation. If the decision maker is unable to select a solution that meets the AVEP requirements and policy restrictions, the AVEP will use a solution that is not fully compliant with the policy generate a new solution on the next iteration.

3.3.6.3 Learn From Experience

The second aspect of learning is learn from experience, where the AVEP modifies its current behavior based on its previous behavior.

The decision maker calculates a score for each solution in the solution space. The score is a weighted sum of the objective values in a single solution, resulting in a single value for each solution in the solution space.

For a solution space, the scoring function normalizes objective values in the solution space so that they are each scaled (0, 1), that is, $F = \{f | 0 \leq f \leq 1\}$, where F is an objective value in the solution space. (This process is not actually performed on the Z values, as the Z objective function is defined such that $Z = \{z | 0 \leq z \leq 1\}$.)

With all the values in the solution space normalized, a single score for a solution [Z, T, B, G] can be generated. Recalling that we wish to minimize the bit error rate and the time of travel along a path, and maximize the packet delivery and target/anti-target value, (14) is a reasonable and simple method of scoring the solutions in the solution space:

$$S = Z - T - B + G \tag{14}$$

where S is the solution score. This score is returned to the AVEP along with the solution itself and the solution parameters, where it is used in the second stage of the learning process. The second stage learning process is designed to provide AVEP learning in the mode of machine learning or cognition. While the decision maker provides decision and action functions, the second stage learning provides context to the decisions. Second stage learning provides the adaptive adaptation that differentiates cognitive systems from simply adaptive systems according to Haigh [46].

The second stage learning process uses the solution score to determine whether to apply the current solution, or use a different solution. The nondominated sort generates a set of nondominated solutions, any of which may be suitable for AVEP operation. The solution score attempts to provide some method of comparing solutions from one iteration to the next. When the decision maker returns a solution and a score, the AVEP controller compares the current score with the score associated with the most recently implemented solution. If the current score is higher than the previous score, or if there is no previous score, then the AVEP implements the current solution. But if the current score is not higher than the previous score, and the previous solution is still valid (that is, if the environment in which the previous solution was generated has not changed), then the AVEP will reuse the previously generated solution. Thus the AVEP can choose to implement a solution returned by the decision maker if the solution provides an improvement in performance as indicated by the solution score, or the AVEP can choose to use a solution that provided satisfactory performance previously.

In addition to allowing the AVEP to incrementally increase the effectiveness of AVEP performance, the second stage learning process provides a level of persistence to the decision making process. The decision maker operates only on the information it has on hand, the meters recorded as the AVEP traverses each path. But the second stage learning provides context for those decisions. The AVEP can compare solutions from different iterations, different points in time, and choose which one to ultimately implement.

However, there is a risk associated with using the previous solution if the current solution does not have a high score. The AVEP reads environmental meters as it traverses a path. If the AVEP continually uses the previous solution, it is continually traversing the same path, and any changes occurring on other paths remain unobserved. Therefore, the AVEP keeps track of the number of times it dispenses with the current solution in favor of the previous solution. When this occurs a certain number of times (currently the threshold is set at 5), the AVEP automatically explores the other paths to record the path meters. After this re-exploration process, the AVEP goes back to generating and implementing solutions, now with up to date information.

4.0 Results and Discussion

This section presents AVEP experimental tests and results. The experiments are divided into two sections, software-based simulation, and robot-deployed live tests.

There is a lack of standardized experimental procedures or metrics for CR. Our work combining CR and AV research further highlights this situation. In order to validate the experimental results, we will start simply and build up. The AVEP objective functions are target/anti-target score (Z), travel time (T), bit error rate (B), and packet delivery (G). We verified that the decision making process works for individual objective functions; that is, we can isolate an individual objective function and show that the decision making process returns a solution that makes sense when the objective function is the only one considered [87]. The next step is to evaluate the results when considering all four objective functions together. We evaluated the full decision making process over many iterations, in scenarios represent static environments, a simple dynamic environment (environment changes once during the scenario), and rapidly changing dynamic environments (values can change dramatically from iteration to iteration).

4.1 Software Simulation

The simulation results are generated in software. We use the same modules for simulation as those deployed on the AVEP. The simulation controller employs the same FSM as the AVEP controller (Section 6.3.4.1.1), but while the AVEP employs actual environmental and RF sensing, the simulation uses a pre-generated set of environmental parameters (X, Y, and Noise) that are fed to the simulation controller during every iteration. The simulation controller feeds the values to the decision maker, which generates a solution space and returns a solution to the simulation controller. The simulation controller also implements the second stage learning process, as discussed in Section 6.3.6.3.

4.1.1 Individual Objective Functions

We first demonstrated the effectiveness of the decision making process in [65]. We isolated a single objective and showed that the system was able to make an appropriate choice, and we reproduce the results here.

The three lines below reproduce the Python dictionaries that store the solution parameters generated during the decision making process.

```
{’dist’:62.0, ..., ’Z’: 0.0, ...}

{’dist’:48.0, ..., ’Z’: 0.40, ...}
```

```
{’dist’:87.5, ..., ’Z’: 0.599, ...}
```

These results were generated when the decision maker was asked to choose a solution that maximizes the Z score. Each dictionary represents a different test bed path, and the RF and MOT parameters the AVEP would implement while traversing the path. Of the three possible paths, the decision making process chose the third, the path with the highest Z score.

When the decision maker was asked to choose a solution that minimized the time, T, the second path was chosen. Clearly, in the absence of any other considerations, the path with the shortest length will result in the shortest travel time.

```
{’dist’:62.0, ’T’:8.967, ..., ’rotor power’: 50.0, ...}

{’dist’:48.0, ’T’:6.942, ..., ’rotor power’: 50.0, ...}

{’dist’:87.5, ’T’:12.655, ..., ’rotor power’: 50.0, ...}
```

Tasked with minimizing the B objective, BER, the decision maker selected a solution that uses high transmit power and low data rate. This is a “pure” RF solution, i.e., the choice of solution in this case requires no consideration of MOT parameters. In this case, the choice of path has no impact, as the solution parameters are the same for each path, and the decision maker selected the first path.

```
{..., ’Rs’:2000.0, ’B’:0.0, ..., ’EIRP’:17.0, ...}

{..., ’Rs’:2000.0, ’B’:0.0, ..., ’EIRP’:17.0, ...}

{..., ’Rs’:2000.0, ’B’:0.0, ..., ’EIRP’:17.0, ...}
```

The final objective function the decision maker optimized in isolation was packet delivery, maximizing G. The decision maker used a high bit rate while driving the rotor power (and therefore the AVEP speed) down to increase the time available to transmit the packets. This combination of RF and MOT parameters, implemented on the longest path in the test bed (the third path) resulted in the best results for G. This clearly shows the coupled nature of RF and MOT parameters, and is a simple example of why we are considering RF and MOT together.

{’dist’:62.0, ’T’:14.5, ’Rs’:57600.0, ..., ’rotor power’: 25.0, ’G’:45}

{’dist’:48.0, ’T’:11.2, ’Rs’:57600.0, ..., ’rotor power’: 25.0, ’G’:35}

{’dist’:87.5, ’T’:20.4, ’Rs’:57600.0, ..., ’rotor power’: 25.0, ’G’:64}

4.1.2 Multi-domain Decision Making in a Static Environment

The next step is to evaluate the full UMDDM process in a static environment, i.e., an environment that does not change from iteration to iteration. The environment for an iteration is established by fixing the meters for each path that the decision maker will use in the decision making process. The parameters used for this analysis are shown in Table 3. The X and Y are values that are reproducible in the test bed, while the Noise value represents the empirically determined value as discussed in Section 6.3.6.2.1.3.

Table 3. Environmental parameters (meters) used in evaluating UMDDM in a static environment.

Path	Targets X	Anti-targets Y	Noise (dBm)
A	5	1	-92
B	3	0	-92
C	5	3	-92

We ran the decision maker through 20 iterations, simulating the choices the AVEP would make if it traveled around the test bed 20 times. The results are shown in Table 4. The table highlights several different aspects of the decision making and learning process. The “Mode” column shows whether the AVEP is actively communicating while traversing its chosen path (“Exploit”), or whether it is exploring the path environment to ensure up-to-date meter values (“Explore”). Notice that the second stage learning process ensures that the AVEP does not spend all its time exploiting the best path at the expense of exploring the other paths. Rather, the second stage learning process ensures that the AVEP does maintain up-to-date meter readings by periodically re-exploring other paths. Also note that the solution scores never decrease. Over the course of an entire operational session, the score increases, indicating that the AVEP is incrementally improving its performance over the long term.

The first three iterations of the operational session are used to explore each of the three paths. Iteration four is the first time the decision maker must generate a solution and parameters to implement. While new solutions are generated each iteration that the AVEP is not exploring, in this scenario, the AVEP implements the new solutions on iterations 4, 5, and 10. Table 5 shows the solutions generated by the decision maker for those iterations. The trade-offs inherent in choosing one nondominated solution over another are clear. From iterations 4 to iteration 5, T is decreased as desired, and Z is increased (also desirable), but at the cost of a decrease in G.

Table 4. Results of UMDDM decision making simulation in a static environment over 20 iterations.

Iteration	Path traversed	Mode	New/Prev solution	Solution score
1	A	Explore	N/A	N/A
2	B	Explore	N/A	N/A
3	C	Explore	N/A	N/A
4	C	Exploit	New solution	0.2319
5	A	Exploit	New solution	0.5874
6	A	Exploit	Prev solution	0.5874
7	A	Exploit	Prev solution	0.5874
8	A	Exploit	Prev solution	0.5874
9	A	Exploit	Prev solution	0.5874
10	B	Exploit	New solution	0.6013
11	B	Exploit	Prev solution	0.6013
12	A	Explore	N/A	N/A
13	C	Explore	N/A	N/A
14	B	Exploit	Prev solution	0.6013
15	B	Exploit	Prev solution	0.6013
16	B	Exploit	Prev solution	0.6013
17	B	Exploit	Prev solution	0.6013
18	B	Exploit	Prev solution	0.6013
19	A	Explore	N/A	N/A
20	C	Explore	N/A	N/A

Table 6 shows the parameters that AVEP uses to implement the solutions shown in Table 4 above. The differences and trade-offs between solutions are more obvious here. The AVEP starts out on the longest path, but as it switches to shorter paths, it increases the bit rate and reduces its movement power slightly to maintain a reasonable value for packet delivery. This scenario is a perfect example of how MOT and RF parameters can be exchanged to ensure mission success.

Table 5. Solutions generated by decision maker on iterations 4, 5, and 10, and the scores associated with each solution.

Iteration	Z	T	B	G	Score
4	0.4000	13.0587	0	31	0.2319
5	0.5999	9.2530	0	29	0.5874
10	0.6000	7.4189	0	24	0.6013

Table 6. Parameters associated with solutions shown in Table 6.

Iteration	Path	Length (m)	R_s (kbps)	EIRP (dBm)	Rotor power
4	C	2.223	9.6	17.0	55
5	A	1.575	57.6	17.0	55
10	B	1.219	125.0	17.0	40

The nondominated sort algorithm generates a front with 154 members (out of a solution space with 1152 members). Using UMDDM, the AVEP uses an optimal solution every time it must make a decision. For comparison, we used Python’s random number generator to generate 50 uniformly distributed samples from the solution space, and none of the randomly selected solutions was present in the nondominated solution set. Clearly UMDDM provides better results than selecting operational parameters at random. UMDDM also implements intelligent adaptation, using the second stage learning, which provides incremental improvement in operational performance over time. This is highlighted by the increasing score, recorded in Table 4 and Table 5.

It should be noted with that with the environmental parameters set as noted in Table 3, and with the available set of knobs, the BER will always be 0. Figure 35 shows that for a wide range of RS values, the B value (BER) is effectively 0 for SNR values greater than 5 dB. If we increase the Noise value in the simulated environment to −82 dBm, this will generate some variability in B values in the solutions space generated by the decision maker. Table 7 shows the results of running the simulation again with the higher noise value. There is nothing significant to observe in this table as compared to Table 4, but we have included the information for the sake of completeness.

Table 8 shows the solutions generated by the decision maker and implemented by the AVEP in the scenario where the simulated environment has noise N = −82 dBm. Note that the higher noise does result in variability in the B values. Again, the trade-offs are clear when choosing one nondominated solution over another. For example, from iteration 6 to 10, the T value worsens (time increases) while the B value improves (BER decreases). From iteration 10 to 14, the T value improves, while the B value worsens. Table 9 shows the parameters used to implement the solutions in this scenario. As in the previous static scenario, the decision maker generates solution with maximum EIRP. The decision maker seeks to minimize B, and this can be done by driving the transmit power up. In this working proof of concept prototype, we implemented only 4 objective functions that clearly show the multi-domain aspects of UMDDM. There are methods to reduce the tendency to drive input parameters to their maximum values, such as including the parameter as an objective function [5], [66] or penalizing solutions that result in high transmit power [67].

Table 7. Results of UMDDM decision making simulation in a static environment with N = –82 dBm.

Iteration	Path traversed	Mode	New/Prev solution	Solution score
1	A	Explore	N/A	N/A
2	B	Explore	N/A	N/A
3	C	Explore	N/A	N/A
4	B	Exploit	New solution	0.4256
5	B	Exploit	Prev solution	0.4256
6	B	Exploit	New solution	0.5295
7	B	Exploit	Prev solution	0.5295
8	B	Exploit	Prev solution	0.5295
9	B	Exploit	Prev solution	0.5295
10	B	Exploit	New solution	0.5726
11	B	Exploit	Prev solution	0.5726
12	A	Explore	N/A	N/A
13	C	Explore	N/A	N/A
14	B	Exploit	New solution	0.5750
15	B	Exploit	New solution	0.5846
16	B	Exploit	Prev solution	0.5846
17	B	Exploit	Prev solution	0.5846
18	B	Exploit	Prev solution	0.5846
19	B	Exploit	Prev solution	0.5846
20	B	Exploit	Prev solution	0.5846

Table 8. Solutions generated by decision maker for simulated environment where N = –82 dBm.

Iteration	Z	T	B	G	Score
4	0.6000	8.2233	0	15	0.4256
6	0.6000	9.4762	0.0209	30	0.5295
10	0.6000	9.8670	1.6111e-5	30	0.5726
14	0.6000	9.4762	5.0631e-4	29	0.5750
15	0.6000	7.4189	5.0631e-4	23	0.5846

Table 9. Parameters associated with solutions shown in Table 8

Iteration	Path	Length (m)	R_s (kbps)	EIRP (dBm)	Rotor power
4	B	1.219	4.8	17.0	35
6	B	1.219	125	17.0	30
10	B	1.219	38.4	17.0	75
14	B	1.219	57.6	17.0	30
15	B	1.219	57.6	17.0	40

4.1.3 Multi-domain Decision Making in a Simple Dynamic Environment

To evaluate UMDDM in a simple dynamic environment, we generated a scenario that changed the parameters of Path B at iteration 10, as noted in Table 10.

Table 10. Environmental parameters (meters) used in evaluating UMDDM in a simple dynamic environment.

Iterations	Path	Targets X	Anti-targets Y	Noise (dBm)
1-10	A	5	1	-82
	B	3	0	-82
	C	5	3	-82
11-20	A	5	1	-82
	B	0	0	-82
	C	5	3	-82

The results of this test scenario are shown in Table 11. The AVEP uses the second stage learning to increase the score on iterations 4 and 6, but the score drops again on iteration 13. This reflects the change in the environment that occurs on iteration 11; however, this change is not registered until iteration 13. During iteration 13, the AVEP traverses path B and updates its internal information. This updated information is incorporated into the learning process, and a new solution space is generated. The selected solution has a lower score than the previously implemented solution, but the AVEP recognizes that the environment in which the previous solution was generated no longer exists, and the previous solution is not relevant. As before, the AVEP periodically re-explores paths that have not been recently traversed, in order to maintain up to date information for effective decision making.

Table 11. Results of UMDDM decision making simulation in a simple dynamic environment

Iteration	Path traversed	Mode	New/Prev solution	Solution score
1	A	Explore	N/A	N/A
2	B	Explore	N/A	N/A
3	C	Explore	N/A	N/A
4	B	Exploit	New solution	0.5211
5	B	Exploit	Prev solution	0.5211
6	B	Exploit	New solution	0.5819
7	B	Exploit	Prev solution	0.5819
8	B	Exploit	Prev solution	0.5819
9	B	Exploit	Prev solution	0.5819
10	B	Exploit	Prev solution	0.5819
11	A	Explore	N/A	N/A
12	C	Explore	N/A	N/A
13	B	Exploit	Prev solution	0.5819
14	A	Exploit	New solution	0.5408
15	A	Exploit	Prev solution	0.5408
16	A	Exploit	Prev solution	0.5408
17	A	Exploit	Prev solution	0.5408
18	A	Exploit	Prev solution	0.5408
19	B	Explore	N/A	N/A
20	C	Explore	N/A	N/A

Table 12 shows the solutions generated by the decision maker and implemented by the AVEP in this scenario, which simulates a slowly changing dynamic environment. Once again, it is possible observe the trade-offs associated with selecting a solution from multiple nondominated solutions. From iteration 4 to iteration 6, the T value is improved, while the B and G values decline (BER increases and packet delivery is reduced). From iteration 6 to iteration 10, the T and B values improve, while the G value again decreases. Table 13 shows the parameters associated with each implemented solution, and a corresponding trade-off between parameters can also be observed.

Table 12. Solutions generated by decision maker in a simple dynamic environment, and the scores associated with each solution.

Iteration	Z	T	B	G	Score
4	0.6000	11.2310	5.1918e-10	31	0.5211
6	0.6000	7.4189	5.0631e-4	28	0.5819
14	0.5999	8.2233	5.1918e-10	25	0.5408

Table 13. Parameters associated with solutions shown in Table 12.

Iteration	Path	Length (m)	R_s (kbps)	EIRP (dBm)	Rotor power
4	B	1.219	19.2	17.0	25
6	B	1.219	57.6	17.0	70
14	A	1.575	19.2	17.0	50

4.1.4 Multi-domain Decision Making in a Highly Dynamic Environment

We created a randomly generated scenario to test the AVEP decision making and learning performance in a highly dynamic environment. Table 14 shows the meters for the first ten iterations of this scenario. This scenario represents situations where the AVEP encounters a rapidly changing environment. Such situations might arise as a result of high speed operation or extremely hostile environments.

Table 14. Environmental parameters (meters) used in evaluating UMDDM in a highly dynamic environment.

Iterations	Path	Targets X	Anti-targets Y	Noise (dBm)
1	A	6	7	-87.1
	B	8	7	-83.1
	C	3	0	-76.7
2	A	3	8	-75.5
	B	10	3	-71.5
	C	1	1	-83.1
3	A	6	4	-90.7
	B	5	6	-68.3
	C	6	8	-98.2
4	A	9	10	-85.5
	B	9	6	-86.5
	C	0	7	-60.11
5	A	9	2	-78.6
	B	5	3	-76.0
	C	9	4	-87.6
6	A	0	5	-77.7
	B	9	6	-77.8
	C	9	2	-98.2
7	A	6	1	-86.9
	B	2	3	-81.8
	C	5	4	-76.9
8	A	0	6	-81.6
	B	1	1	-82.3
	C	3	5	-76.9
9	A	1	6	-80.0
	B	7	5	-69.5
	C	10	2	-86.8
10	A	4	2	-81.6
	B	5	5	-91.7
	C	10	7	-85.0

The results of this test scenario are shown in Table 15. In this scenario where the environment changes dramatically with every iteration, the AVEP implements a new solution every iteration and is unable to use previous solutions in an attempt to increase its performance from one iteration to the next, as indicated by the solution score. While the AVEP does seek to maintain high performance by selecting nondominated solutions from the solution space generated by the decision maker, the AVEP does not "chase" a single best solution. Newman et al. have shown that a system that changes operationally parameters in response to environmental cues too readily can be "herded" or exploited, guided to operate in a manner beneficial to external agents [68]. As the AVEP makes decisions based only on the information is has gathered, it is possible that in a highly dynamic environment, the AVEP will miss some opportunities, such as the extremely beneficial environment offered by path C on iteration 9. On the other hand, the AVEP continues to operate even while in a hostile environment, such as that of path C on iteration 4. The AVEP seeks to improve performance after every iteration, whether in a favorable or hostile environment.

Table 15. Results of UMDDM decision making simulation in a highly dynamic environment over 10 iterations.

Iteration	Path traversed	Mode	New/Prev solution	Solution score
1	A	Explore	N/A	N/A
2	B	Explore	N/A	N/A
3	C	Explore	N/A	N/A
4	C	Exploit	New solution	0.0002
5	A	Exploit	New solution	-0.0372
6	A	Exploit	New solution	0.7539
7	B	Exploit	New solution	0.7901
8	A	Exploit	New solution	-0.1051
9	A	Exploit	New solution	-0.2588
10	B	Exploit	New solution	-0.2849

Table 16 shows the solutions generated by the decision maker and implemented by the AVEP in this scenario, which simulates a rapidly changing dynamic environment. Once again, it is possible observe the trade-offs associated with selecting a solution from multiple nondominated solutions. This scenario presents instances where selected solutions are not fully policy compliant ($Z < 0.1$ for iterations 4, 5, 6, 8, and 9). While the decision maker does generate every possible solution in the solution space, it does not exhaustively search the solution space for a solution. Instead, the decision maker uses the nondominated sort to select a suitable subsection of the entire population, and makes an effort to choose a solution that satisfies policy. But if the decision maker is unable to find a compliant solution, the decision maker will return a solution that is not compliant with policy. This avoids a deadlock state, searching for a nondominated solution that satisfies policy which may not exist. On the other hand, the decision maker may miss solutions that are compliant and return a non-complaint solution. We have chosen to accept this trade off, as the result is a system that is robust and stable in the face of hostile environments with limited solution possibilities: implementing a non-compliant solution ensures that AVEP operation actually continues, and with limited delay.

In this highly dynamic environment, the second stage learning process is unable to improve performance from one iteration to the next through the use of previously implemented solutions; in this scenario, previous solutions are never applicable as the environment along every path changes with every iteration. In this case, the AVEP is intelligent enough not to use previous results while the environment is rapidly changing. Yet if the environment were to settle, the AVEP would then apply the second stage learning process and seek iterative improvement using previously implemented solutions. Table 17 shows the parameters that correspond to the solutions generated and implemented in this scenario.

Table 16. Solutions generated by decision maker in a highly dynamic environment, and the scores associated with each solution.

Iteration	Z	T	B	G	Score
4	0.0	17.9867	0	58	-0.1208
5	0.0	12.2401	0	37	-0.0228
6	0.0	13.8606	1.6038e-5	44	0.2599
7	1.0	9.8670	3.3715e-4	18	-0.0372
8	0.0	14.5068	8.8464e-11	48	-0.0393
9	0.0	12.7449	0	24	-0.0057
10	1.0	9.8670	2.2734e-7	13	-0.1060

Table 17. Parameters associated with solutions shown in Table 16.

Iteration	Path	Length (m)	R_s (kbps)	EIRP (dBm)	Rotor power
4	C	2.223	125.0	17.0	75
5	A	1.575	38.4	17.0	30
6	A	1.575	125.0	17.0	80
7	B	1.219	4.8	17.0	75
8	A	1.575	57.6	17.0	25
9	A	1.575	4.8	17.0	75
10	A	1.575	2.4	17.0	75

4.2 Live Tests

In this section, we present the results from the live tests conducted using the AVEP on the test bed. We will evaluate the full decision making process over many iterations, in scenarios representing static environments, a simple dynamic environment (environment changes once during the scenario), and rapidly changing dynamic environments (values can change dramatically from iteration to iteration). Figure 37 shows the AVEP in operation during one of the live test experiments.

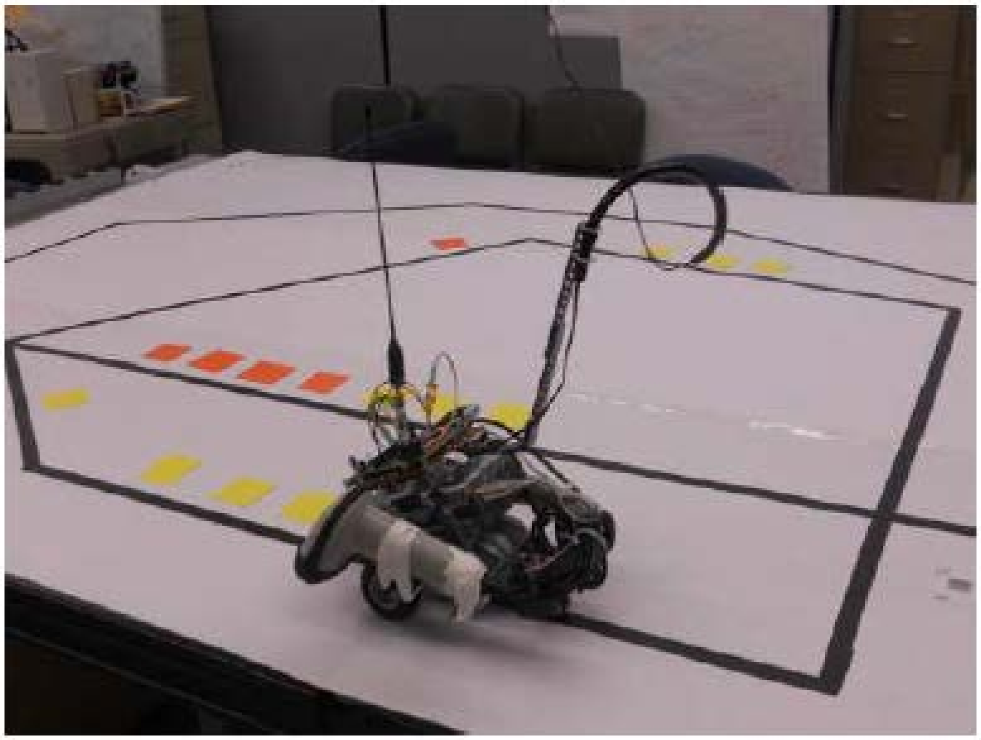


Figure 37. AVEP in operation during a live test experiment.

4.2.1 Live Multi-domain Decision Making in a Static Environment

This scenario mirrors the software simulation of the full UMDDM process in a static environment. The environment for an iteration is established by fixing the meters for each path that the decision maker will use in the decision making process. The parameters used for this analysis are shown in Table 18. Notice that these are the same values as in Table 3. The X and Y are values fixed by using pieces of colored paper on a given test bed path. The Noise value is a close approximation of the Noise value seen by the receiver while operating in the test bed.

Table 18. Environmental parameters (meters) used in evaluating live UMDDM in a static environment.

Path	Targets X	Anti-targets Y	Noise (dBm)
A	5	1	-92
B	3	0	-92
C	5	3	-92

Test bed experiments were run in real time in the test bed, and as a result took much longer to complete than the software simulations. We ran the test bed experiments for ten iterations, and changed the threshold that controls the re-exploration of alternative paths from 5 to 3. In this manner, we still show all the details of the decision making and learning processes, while completing the experiments in a reasonable time. Table 19 shows the results from a live test bed experiment in a static environment.

Table 19. Results of live UMDDM in a static environment over 10 iterations.

Iteration	Path traversed	Mode	New/Prev solution	Solution score
1	A	Explore	N/A	N/A
2	B	Explore	N/A	N/A
3	C	Explore	N/A	N/A
4	C	Exploit	New solution	0.3648
5	A	Exploit	New solution	0.5713
6	A	Exploit	Prev solution	0.5713
7	A	Exploit	Prev solution	0.5713
8	A	Exploit	Prev solution	0.5713
9	B	Explore	N/A	N/A
10	C	Explore	N/A	N/A

During live tests, the AVEP uses the first three iterations to explore the environment, recording the number of targets and anti-targets along each path, and recording the noise level observed by the RFIC while traversing each path. As with the software simulations above, the decision maker generates a new solution on iteration 4. On iteration 5, the AVEP again uses a new solution, but continues to use that solution on iterations 6, 7, and 8. From iteration 4 to 5, the score increases, indicating that the AVEP is incrementally improving its performance over the long term. The second stage learning process ensures that the AVEP maintains up to date information. After three iterations using the same previous solution, the AVEP switches from "Exploit" to "Explore" mode and re-explores paths B and C to obtain up to date information.

Table 20 shows the solutions generated by the decision maker and implemented by the AVEP on iterations 4 and 5 during the live experiments in a static environment. As mentioned previously, the noise in the test bed environment is approximately N = −92 dBm as observed by the RFIC. At this level and for a wide range of R_S values, the B value (BER) is effectively 0 for SNR values greater than 5 dB. This can be observed in the table.

The trade-offs associated with selecting a solution among nondominated candidates is again observable. From iteration 4 to 5, the Z and T values at the expense of reduced packet delivery. Table 21 shows the corresponding trade-offs in parameters: R_s is reduced, while the AVEP rotor power is increased.

Table 20. Solutions generated by decision maker during live tests in a static environment.

Iteration	Z	T	B	G	Score
4	0.4000	14.9903	0	46	0.3648
5	0.5999	9.5827	0	29	0.5713

Table 21. Parameters associated with solutions shown in Table 20

Iteration	Path	Length (m)	R_s (kbps)	EIRP (dBm)	Rotor power
4	C	2.223	57.6	17.0	35
5	A	1.575	38.4	17.0	55

By transmitting a broadband signal on the frequency that the AVEP uses, and by transmitting at very low power, it is possible to simulate a higher noise floor in the test bed environment. Using this method, we were able to raise the noise to approximately N = −83 dBm, which resulted in some variability in the B values generated by the decision maker. We re-ran the static environment test bed experiment using the broadband signal to raise the noise floor. The results are shown in Table 22.

Table 22. Results of live UMDDM in a static environment over 10 iterations using higher noise floor.

Iteration	Path traversed	Mode	New/Prev solution	Solution score
1	A	Explore	N/A	N/A
2	B	Explore	N/A	N/A
3	C	Explore	N/A	N/A
4	A	Exploit	New solution	0.3169
5	B	Exploit	New solution	0.4665
6	A	Exploit	New solution	0.9670
7	A	Exploit	Prev solution	0.9670
8	A	Exploit	Prev solution	0.9670
9	A	Exploit	Prev solution	0.9670
10	B	Explore	N/A	N/A

Table 23 shows the solutions generated by the decision maker in this re-run test bed experiment. Note the value of B on iteration 6. This is a result of the increased noise floor, and the higher data rate shown in Table 24. We conducted the remaining experiments in this section using the broadband signal to increase the noise floor observed the RFIC.

Table 23. Solutions generated by decision maker during live tests in a static environment with higher noise floor.

Iteration	Z	T	B	G	Score
4	0.5999	13.8606	0	26	0.3169
5	0.6000	9.8670	0	23	0.4665
6	0.5999	12.2401	1.7094e-4	39	0.9670

Table 24. Parameters associated with solutions shown in Table 23.

Iteration	Path	Length (m)	R_s (kbps)	EIRP (dBm)	Rotor power
4	A	1.575	4.8	17.0	80
5	B	1.219	9.6	17.0	75
6	A	1.575	125.0	17.0	30

4.2.2 Live Multi-domain Decision Making in a Simple Dynamic Environment

Having now shown that the AVEP can learn its surroundings by exploring the paths during test bed experiments, we intentionally bypassed the initial exploration process for this experiment. By pre-loading the AVEP with the information that would have been learned during the initial path explorations, we can focus on the showing how the AVEP uses decision making and the second stage learning process during the test bed experiment.

This experiment shows the operation of the AVEP in a simple dynamic environment, as in Section 7.1.3. Table 25 shows the changes in the number of target and anti-targets pieces on paths A and B from iterations 1-5 to iterations 6-10.

Table 25. Environmental parameters (meters) used in evaluating UMDDM in a simple dynamic environment during a live experiment.

Iterations	Path	Targets X	Anti-targets Y	Noise (dBm)
1-5	A	5	1	-82
	B	3	0	-82
	C	5	3	-82
6-10	A	3	1	-82
	B	0	0	-82
	C	5	3	-82

Table 26 shows the experimental results for this live test bed experiment in a simple dynamic environment. As in the simulations, the AVEP seeks to maintain or increase performance, as indicated by the solution score. In the simulations, the only time that the score decreased was when the environment changed from one iteration to the next. Note that the score does drop on iteration 3, when the environment has not in fact changed. This is due to sensor jitter; during live tests, the chassis mounted sensors will occasionally register more targets or anti-targets than actually exist due to the motion of the AVEP itself. This could be fixed with a more sophisticated sensing algorithm, but that is outside the scope of this work. On iteration 6, the target and anti-target values do change, and this is correctly observed by the AVEP. On iteration 6, the AVEP traverse path B using a previously developed solution, but observes that the environment has changed and discards the solution. The AVEP selects path A for the next iteration, and develops a solution based on expired data. As soon as the AVEP traverses path A, it observers that path A has also changed, and discards the expired solution for path A as well. On iteration 8, the AVEP implements a solution that corresponds to the current path environment, and uses this solution for the remaining iterations of the experiment.

Table 26. Results of live UMDDM in a simple dynamic environment over 10 iterations in a live experiment.

Iteration	Path traversed	Mode	New/Prev solution	Solution score
1	A	Exploit	New solution	0.5874
2	A	Exploit	New solution	0.6301
3	C	Exploit	New solution	-0.0057
4	A	Exploit	New solution	0.2850
5	B	Exploit	New solution	0.5981
6	B	Exploit	Prev solution	0.5981
7	A	Exploit	New solution	0.2630
8	A	Exploit	New solution	0.1286
9	A	Exploit	Prev solution	0.1286
10	A	Exploit	Prev solution	0.1286

Table 27 shows the solutions generated by the decision maker and implemented by the AVEP during the live experiment in a simple dynamic environment. As shown above, this table also displays the results of the changing environment and the solutions generated in iterations 7 and 8 to accommodate the changes. Table 28 shows the parameters associated with each solution.

Table 27. Solutions generated by decision maker during live tests in a simple dynamic environment.

Iteration	Z	T	B	G	Score
1	0.5999	14.5067	1.7094e-4	46	0.5874
2	0.5999	8.9673	0	21	0.6301
3	0.4000	13.8907	0	18	-0.0057
4	0.5999	13.8606	0.1709e-4	44	0.2850
5	0.6000	9.0251	0.1709e-4	29	0.5981
7	0.5999	10.6217	0	12	0.2630
8	0.5999	8.9673	0	11	0.1286

Table 28. Parameters associated with solutions shown in Table 27.

Iteration	Path	Length (m)	R_s (kbps)	EIRP (dBm)	Rotor power
1	A	1.575	125.0	17.0	25
2	A	1.575	9.6	17.0	50
3	C	2.223	2.4	17.0	60
4	A	1.575	125.0	17.0	80
5	B	1.219	125.0	17.0	70
7	A	1.575	2.0	17.0	35
8	A	1.575	2.4	17.0	50

4.2.3 Live Multi-domain Decision Making in a Highly Dynamic Environment

For this live experiment, we again created a randomly generated scenario to test the AVEP decision making and learning performance in a highly dynamic environment. Using the randomly generated environmental parameters, we were able to implement the experiment by changing the target and anti-target parameters along each path between every iteration of the experiment. While we did conduct this experiment while transmitting a broadband signal to raise the noise floor, further refinements to the test bed's RF environment are not practical. Table 29 shows the meters for ten iterations.

Table 29. Environmental parameters (meters) used in evaluating UMDDM in a highly dynamic environment during a live experiment.

Iterations	Path	Targets X	Anti-targets Y	Noise (dBm)
1	A	1	3	-82
	B	1	2	-82
	C	3	3	-82
2	A	0	3	-82
	B	0	1	-82
	C	3	5	-82
3	A	0	5	-82
	B	5	0	-82
	C	2	2	-82
4	A	1	5	-82
	B	3	4	-82
	C	0	2	-82
5	A	5	2	-82
	B	3	5	-82
	C	3	4	-82
6	A	3	3	-82
	B	1	4	-82
	C	1	4	-82
7	A	2	3	-82
	B	3	4	-82
	C	0	0	-82
8	A	4	4	-82
	B	1	2	-82
	C	2	0	-82
9	A	2	3	-82
	B	4	1	-82
	C	3	2	-82
10	A	5	1	-82
	B	3	4	-82
	C	3	1	-82

The results of this live test experiment are shown in Table 30. As in the highly dynamic tests scenario simulation, in this experiment where the environment changes dramatically with every iteration, the AVEP implements a new solution every iteration and is unable to use previous solutions in an attempt to increase its performance from one iteration to the next, as indicated by the solution score. The AVEP makes decisions based only on the information is has gathered. While it is possible that in a highly dynamic environment the AVEP will miss some opportunities, in this experiment the AVEP manages to exploit some of the better path environments, such as path B on iteration 3, path C on iteration 8, and path A on iteration 10.

Table 30. Results of live UMDDM in a simple dynamic environment over 10 iterations in a live experiment.

Iteration	Path traversed	Mode	New/Prev solution	Solution score
1	C	Exploit	New solution	0.2933
2	A	Exploit	New solution	0.5864
3	B	Exploit	New solution	0.3793
4	B	Exploit	New solution	0.5709
5	B	Exploit	New solution	-0.0290
6	C	Exploit	New solution	-0.1066
7	C	Exploit	New solution	-0.4853
8	C	Exploit	New solution	-0.2433
9	C	Exploit	New solution	0.1396
10	A	Exploit	New solution	-0.2701

Table 31 shows the solutions generated by the decision maker and implemented by the AVEP in this experiment. As mentioned in previous experiment, we transmitted a broadband signal to increase the noise as observed by the RFIC. Even with this signal raising the noise floor, there is very little variability in the B values of the solutions generated by the decision maker. This is due to the nature of the B objective function. Figure 6.8 shows that for a wide range of RS values, the B value (BER) is effectively 0 for SNR values greater than 5 dB. In live experiments, it is difficult to fix the RF environment, as opposed to simulations, where the noise value can be fixed to a specific value to ensure that the decision maker generates solutions with variability in the B values. Still, it is possible observe the trade-offs made between solutions from one iteration to the next. For example, from iteration 5 to iteration 6, the Z and T values degrade (Z goes down, T goes up), but the G value improves by more than double. Table 32 shows the parameters associated with the solutions generated by the decision maker and implemented by the AVEP in this live experiment.

Table 31. Solutions generated by decision maker in a highly dynamic environment during a live experiment.

Iteration	Z	T	B	G	Score
1	0.4000	16.4519	0	46	0.2933
2	0.5999	9.5827	1.5031e-8	30	0.5864
3	0.6000	7.6200	0	10	0.3793
4	0.6000	7.4189	0	22	0.5709
5	0.6000	7.4189	0	22	-0.0290
6	0.4000	16.4519	0	46	-0.1066
7	0.4000	16.4519	0	21	-0.4853
8	0.4000	19.5613	0	47	-0.2433
9	0.4000	17.9867	0	54	0.1396
10	0.5999	9.2530	0	12	-0.2701

Table 32. Parameters associated with solutions shown in Table 31.

Iteration	Path	Length (m)	R_s (kbps)	EIRP (dBm)	Rotor power
1	C	2.223	192.0	17.0	70.0
2	A	1.575	57.6	17.0	40.0
3	B	1.219	2.4	17.0	60.0
4	B	1.219	38.4	17.0	40.0
5	B	1.219	38.4	17.0	40.0
6	C	2.223	19.2	17.0	70.0
7	C	2.223	2.4	17.0	70.0
8	C	2.223	9.6	17.0	80.0
9	C	2.223	38.4	17.0	75.0
10	A	1.575	2.4	17.0	55.0

5.0 Conclusions

This research started several years ago with a simple observation, namely that CRs and AVs perform similar tasks, albeit in different domains:

- Analyze their environment,
- Make and execute a decision,
- Evaluate the result (learn from experience), and
- Repeat as required.

This observation led to trying to combine CR and AV intelligence into a single intelligent agent, with the ability to leverage flexibility in the RF and motion (MOT) domains. We call this idea unified multi-domain decision making (UMDDM).

This report has presented our work on UMDDM, the development of UMDDM algorithms and the implementation of the AVEP test platform, a working proof of concept prototype that deploys UMDDM on a live system.

6.0 Recommendations

In this report, we have described the rationale for and the practical development of a proof-of-concept autonomous vehicle that is able to explore and learn a multidimensional environment that combines changing RF, location, and mission data and to optimize its mission performance intelligently. So far as we are aware, this is the first successful demonstration of its kind. While we describe a wheeled vehicle rather than an aircraft, we believe that the electronic hardware and software at its heart are suitable for installation and testing in a small UAV of the type that AFRL flies. This is the logical next phase of the work, and we strongly recommend that it be done.

7.0 References

[1] T. W. Rondeau and C. W. Bostian, *Artificial Intelligence in Wireless Communications*, 1st ed. Boston: Artech House Publishers, 2009.

[2] N. Wiener, *Cybernetics; or, Control and communication in the animal and the machine*. New York: J. Wiley, 1948.

[3] D. C. Conner, "Sensor Fusion, Navigation, and Control of Autonomous Vehicles," Masters Thesis, Virginia Polytechnic Institute and State University, 2000.

[4] C. J. Rieser, T. W. Rondeau, C. Bostian, W. R. Cyre, and T. M. Gallagher, "Cognitive radio engine based on genetic algorithms in a network," U.S. Patent 728997230-Oct-2007.

[5] T. W. Rondeau, "Application of Artificial Intelligence to Wireless Communications," PhD Dissertation, Virginia Polytechnic Institute and State University, 2007.

[6] M. Angermann, M. Frassl, and M. Lichtenstern, "Autonomous Formation Flying of Micro Aerial Vehicles for Communication Relay Chains," presented at the ION International Technical Meeting, San Diego, CA, 2011.

[7] Y. Mostofi, "Communication-aware motion planning in fading environments," in *IEEE International Conference on Robotics and Automation, 2008. ICRA 2008*, 2008, pp. 3169 –3174.

[8] A. Amanna, M. Gadhiok, M. J. Price, J. H. Reed, W. P. Siriwongpairat, and T. K. Himsoon, "Railway Cognitive Radio," *IEEE Vehicular Technology Magazine*, vol. 5, no. 3, pp. 82–89, Sep-2010.

[9] M. Di Felice, R. Doost-Mohammady, K. R. Chowdhury, and L. Bononi, "Smart Radios for Smart Vehicles: Cognitive Vehicular Networks," *IEEE Vehicular Technology Magazine*, vol. 7, no. 2, pp. 26 –33, Jun. 2012.

[10] D. Bartholomew, *Measuring intelligence : facts and fallacies*. Cambridge, UK; New York: Cambridge University Press, 2004.

[11] M. D. Silvius, "Building a Dynamic Spectrum Access Smart Radio With Application to Public Safety Disaster Communications," PhD Dissertation, Virginia Polytechnic Institute and State University, Blacksburg, Va, 2009.

[12] T. R. Newman, B. A. Barker, A. M. Wyglinski, A. Agah, J. B. Evans, and G. J. Minden, "Cognitive engine implementation for wireless multicarrier transceivers," *Wirel. Commun. Mob. Comput.*, vol. 7, no. 9, pp. 1129–1142, Nov. 2007.

[13] Y. Zhao, S. Mao, J. O. Neel, and J. H. Reed, "Performance evaluation of cognitive radios: Metrics, utility functions, and methodology," *Proceedings of the IEEE*, vol. 97, no. 4, pp. 642–659, 2009.

[14] C. B. Dietrich, E. W. Wolfe, and G. M. Vanhoy, "Cognitive radio testing using psychometric approaches: applicability and proof of concept study," *Analog Integrated Circuits and Signal Processing*, pp. 1–10, 2012.

[15] N. J. Kaminski, "Performance Evaluation of Cognitive Radios," Masters Thesis, Virginia Polytechnic Institute and State University, Blacksburg, Va, 2012.

[16] L. Techy, D. G. Schmale III, and C. A. Woolsey, "Coordinated aerobiological sampling of a plant pathogen in the lower atmosphere using two autonomous unmanned aerial vehicles," *Journal of Field Robotics*, vol. 27, no. 3, pp. 335–343, May 2010.

[17] DARPA, "Urban Challenge Rules." 2007.

[18] RoboCup, "RoboCup Soccer Humanoid League Rules and Setup." 2012.

[19] DARPA, "DARPA Robotics Challenge - DARPA-BAA-12-39 - Federal Business Opportunities: Opportunities," 2012. [Online]. Available: https://www.fbo.gov/index?s=opportunity&mode=form&id=ee8e770bcfe1fe217472342c67d6bd5a. [Accessed: 13-Oct-2012].

[20] Hope Microelectronics Co., Ltd, “RFM22B FSK transceiver - FSK Modules - HOPE Microelectronics,” 2012. [Online]. Available: http://www.hoperf.com/rf_fsk/fsk/RFM22B.htm. [Accessed: 24-Aug-2012].
[21] Tin Can Tools, “Tin Can Tools :: All Products :: Trainer-xM Board,” 2012. [Online]. Available: http://www.tincantools.com/product.php?productid=16151&cat=0&page=2&featured. [Accessed: 18-Oct-2012].
[22] The LEGO Group, “LEGO.com MINDSTORMS : Home,” 2012. [Online]. Available: http://mindstorms.lego.com/en-us/Default.aspx. [Accessed: 17-Oct-2012].
[23] “Hagisonic StarGazer Robot Localization System - RobotShop,” 2012. [Online]. Available: http://www.robotshop.com/hagisonic-stargazer-localization-system-2.html. [Accessed: 26-Nov-2012].
[24] “The Ångström Distribution | Embedded power,” 2012. [Online]. Available: http://www.angstrom-distribution.org/. [Accessed: 18-Oct-2012].
[25] “BeagleboardRevCValidation - beagleboard - Beagle Board Diagnostic Tools and Procedure - Low-cost, low-power single-board computer - Google Project Hosting,” 2012. [Online]. Available: http://code.google.com/p/beagleboard/wiki/BeagleboardRevCValidation. [Accessed: 18-Oct-2012].
[26] “Main Page - Openembedded.org,” 2012. [Online]. Available: http://www.openembedded.org/wiki/Main_Page. [Accessed: 18-Oct-2012].
[27] A. S. Fayez, N. J. Kaminski, A. R. Young, and C. W. Bostian, “Embedded SDR System Design Case Study: An Implmentation Perspective,” presented at the SDR Forum, Washington, D.C, 2012.
[28] Linaro, “Linaro: open source software for ARM SoCs,” 2012. [Online]. Available: http://www.linaro.org/. [Accessed: 18-Oct-2012].
[29] “ARM - Ubuntu Wiki,” 2012. [Online]. Available: https://wiki.ubuntu.com/ARM. [Accessed: 18-Oct-2012].
[30] Python Software Foundation, “Python Programming Language – Official Website,” *Python Programming Language - Official Website*, 2012. [Online]. Available: http://www.python.org/. [Accessed: 16-Oct-2012].
[31] A. R. Young, N. J. Kaminski, A. S. Fayez, and C. W. Bostian, “CSERE (Cognitive System Enabling Radio Evolution): A Modular and User-Friendly Cognitive Engine,” presented at the IEEE DySPAN, Bellevue, WA, in press.
[32] T. W. Rondeau, “GNU Radio - WikiStart - gnuradio.org,” *GNU Radio - WikiStart - gnuradio.org*, 2012. [Online]. Available: http://gnuradio.org/redmine/projects/gnuradio/wiki. [Accessed: 16-Oct-2012].
[33] Baigung, “RFM22B transciever simple example in FIFO mode - Application Notes - HOPE Microelectronics,” 2011. [Online]. Available: http://www.hoperf.com/docs/guide/185.htm. [Accessed: 17-Oct-2012].
[34] C. W. Bostian, Blacksburg, VA, private communication, 20-Aug-2011.
[35] “Scan Codes Demystified.” [Online]. Available: http://www.quadibloc.com/comp/scan.htm. [Accessed: 30-Sep-2012].
[36] IETF, “RFC: 793 Transmission Control Protocol.” 1981.
[37] F. Ge, Q. Chen, Y. Wang, C. W. Bostian, T. W. Rondeau, and B. Le, “Cognitive radio: from spectrum sharing to adaptive learning and reconfiguration,” in *Aerospace Conference, 2008 IEEE*, 2008, pp. 1–10.
[38] B. Le, F. A. G. Rodriguez, Q. Chen, B. P. Li, F. Ge, M. ElNainay, T. W. Rondeau, and C. W. Bostian, “A public safety cognitive radio node,” in *2007 SDR Forum Technical Conference, Denver, CO*, 2007.
[39] S. S. Epp, *Discrete mathematics with applications*. Belmont, CA: Thomson-Brooks/Cole, 2004.
[40] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[41] S. Koenig and M. Likhachev, “D* Lite,” in *Proceedings of the national conference on artificial intelligence*, 2002, pp. 476–483.

[42] C. L. Hwang, *Multiple objective decision making, methods and applications: a state-of-the-art survey*. Berlin ; New York: Springer-Verlag, 1979.

[43] A. He, K. K. Bae, T. R. Newman, J. Gaeddert, K. Kim, R. Menon, L. Morales-Tirado, J. J. Neel, Y. Zhao, J. H. Reed, and others, “A survey of artificial intelligence for cognitive radios,” *Vehicular Technology, IEEE Transactions on*, vol. 59, no. 4, pp. 1578–1592, 2010.

[44] Oxford English Dictionary, “‘learn, v.’.,” 2012. [Online]. Available: http://oed.com/viewdictionaryentry/Entry/106716.

[45] P. McCorduck, *Machines Who Think: A Personal Inquiry into the History and Prospects of Artificial Intelligence*, 2nd ed. A K Peters/CRC Press, 2004.

[46] K. Z. Haigh, “Can Artificial Intelligence meet the Cognitive Networking Challenge?,” presented at the AFRL Cognitive RF Workshop, Dayton, OH, 2011.

[47] J. Mitola III, *Cognitive Radio An Integrated Agent Architecture for Software Defined Radio*. 2000.

[48] L. Doyle, *Essentials of Cognitive Radio*. Cambridge: Cambridge University Press, 2009.

[49] L. Thiele, K. Miettinen, P. J. Korhonen, and J. Molina, “A Preference-Based Evolutionary Algorithm for Multi-Objective Optimization,” *Evolutionary Computation*, vol. 17, no. 3, pp. 411–436, Sep. 2009.

[50] J. Arifovic, “Genetic algorithm learning and the cobweb model,” *Journal of Economic Dynamics and Control*, vol. 18, no. 1, pp. 3–28, Jan. 1994.

[51] K. M. Nelson, “Applications of evolutionary algorithms in mechanical engineering,” 1997.

[52] D. Sahoo, S. C. Rai, and S. Pradhan, “Threshold Cryptography #x00026; Genetic Algorithm Based Secure Key Exchange for Mobile Hosts,” in *Advance Computing Conference, 2009. IACC 2009. IEEE International*, 2009, pp. 1297 –1302.

[53] J. F. Kennedy, R. C. Eberhart, and Y. Shi, *Swarm intelligence*. San Francisco: Morgan Kaufmann Publishers, 2001.

[54] E. Zitzler and L. Thiele, “Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach,” *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 4, pp. 257 –271, Nov. 1999.

[55] A. Rapoport, *Decision theory and decision behaviour*, 2nd rev. ed. Basingstoke: Macmillan, 1998.

[56] “Scientific Computing Tools For Python — Numpy,” 2012. [Online]. Available: http://numpy.scipy.org/. [Accessed: 20-Nov-2012].

[57] B. Sklar, *Digital Communications: Fundamentals and Applications*, 2nd ed. Upper Saddle River, N.J: Prentice Hall, 2001.

[58] T. Hooper, “Communication Systems Modeling with TIMS: Volume D1 Fundamental Digital Experiments.” Emona Instruments Pty Ltd, 2012.

[59] “Question about bandwidth of FSK,” 2009. [Online]. Available: http://www.edaboard.com/thread24570.html. [Accessed: 21-Nov-2012].

[60] T. Pratt, C. W. Bostian, and J. E. Allnutt, *Satellite Communications*, 2nd ed. Wiley, 2002.

[61] A. Leon-Garcia and I. Widjaja, *Communication networks : fundamental concepts and key architectures*. Boston: McGraw-Hill, 2004.

[62] N. Srinivas and K. Deb, “Muiltiobjective Optimization Using Nondominated Sorting in Genetic Algorithms,” *Evolutionary Computation*, vol. 2, no. 3, pp. 221–248, Sep. 1994.

[63] C. Shi, M. Chen, and Z. Shi, “A Fast Nondominated Sorting Algorithm,” in *International Conference on Neural Networks and Brain, 2005. ICNN B ’05*, 2005, vol. 3, pp. 1605 –1610.

[64] B. A. Fette, Ed., *Cognitive Radio Technology*, 1st ed. Newnes, 2006.

[65] A. R. Young and C. W. Bostian, “A Low-cost Cognitive Radio for UAVs that Implements Multi-Domain Decision Making,” presented at the 2012 Cognitive RF Workshop, Kirtland Air Force Base, Albuquerque, NM, 2012.

[66] D. Goodman and N. Mandayam, “Power control for wireless data,” *IEEE Personal Communications*, vol. 7, no. 2, pp. 48 –54, Apr. 2000.

[67] R. Kazemi, R. Vesilo, E. Dutkiewicz, and G. Fang, “Inter-network interference mitigation in Wireless Body Area Networks using power control games,” in *2010 International Symposium on Communications and Information Technologies (ISCIT)*, 2010, pp. 81 –86.

[68] T. R. Newman, T. C. Clancy, M. McHenry, and J. H. Reed, “Case Study: Security Analysis of a Dynamic Spectrum Access Radio System,” in *2010 IEEE Global Telecommunications Conference (GLOBECOM 2010)*, 2010, pp. 1 –6.

8.0 List of Acronyms

ABM: agent based modeling.

AFRL: Air Force Research Lab.

AI: artificial intelligence.

API: application programming interface.

ASIC: application-specific integrated circuit.

AV: autonomous vehicle.

AVEP: autonomous vehicle experimental platform.

AWGN: additive white Gaussian noise.

BAA: broad agency announcement.

BER: bit error rate.

CAM: communication-aware motion.

CE: cognitive engine.

CN: cognitive network.

CNI: Communication Networks Institute.

CORNET: Cognitive Radio Network Testbed.

CR: cognitive radio.

CRC: cyclic redundancy check.

CRE: cognitive radio engine.

CRS: cognitive radio system.

CSERE: Cognitive System Enabling Radio Evolution.

CW: continuous wave.

CWT: Center for Wireless Telecommunications.

DARPA: Defense Advanced Research Projects Agency.

DGC: DARPA Grand Challenge.

DRC: DARPA Robotics Challenge.

DS: dynamic spectrum.

DSA: dynamic spectrum access.

DSP: digital signal processing.

DSS: dynamic spectrum sharing.

DUC: DARPA Urban Challenge.

EIRP: equivalent isotropically radiated power.

EM: electromagnetic radiation.

FIFO: first in first out.

FSK: frequency shift keying.

FSM: finite state machine.

GA: genetic algorithm.

GFSK: Gaussian frequency shift keying.

GPIO: general purpose I/O.

GPS: global positioning system.

I/O: input/output.

I2C: inter-integrated circuit.

LBT: listen before talk.

MAC: media access control.

MAV: micro air vehicle.

MCDM: multiple criteria decision making.

MCU: microcontroller unit.

MDDM: multi-domain decision making.

MDF: mission data file.

MoE: measure of effectiveness.

MOO: multi-objective optimization.

MOT: motion.

MUAV: micro unmanned aerial vehicle.

NAR: Node A radio.

NBR: Node B radio.

NSGA: nondominated sorting genetic algorithm.

ODA: observe, decide, and act.

OE: Open Embedded.

OOK: on-off keying.

OS: operating system.

OTA: over the air.

P/MAC: position/motion-aware communication.

PDS: path data structure.

PHY: physical layer.

POMDP: partially observed Markov decision process.

PU: primary user.

QoS: quality of service.

RCR: railway cognitive radio.

RF: radio frequency.

RFIC: radio frequency integrated circuit.

RNDF: route network definition file.

RSSI: received signal strength indicator.
RX: receive.
SDR: software defined radio.
SNR: signal-to-noise ratio.
SPI: serial peripheral interface.
SU: secondary user.
T/R: transmit/receive.
TCP: transmission control protocol.
TX: transmit.
UAV: unmanned aerial vehicle.
UGV: unmanned ground vehicle.
UMDDM: unified multi-domain decision making.
USB: universal serial bus.
USRP: Universal Software Radio Peripheral.
USV: unmanned surface vehicle.
V2I: vehicle-to-infrastructure.
V2V: vehicle-to-vehicle.
VN: vehicular network.
VT: Virginia Tech.
WARP: Wireless Open-Access Research Platform.
WNaN: Wireless Network after Next.
XG: neXt Generation.